CHAPTER 9

MATLAB Graphics

THE OBJECTIVE OF THIS CHAPTER IS TO INTRODUCE YOU TO:

- MATLAB's high-level 2-D and 3-D plotting facilities
- Handle Graphics
- Editing plots
- Animation
- Saving and exporting graphs
- Color, lighting and the camera

A picture, it is said, is worth a thousand words. MATLAB has a powerful graphics system for presenting and visualizing data, which is reasonably easy to use. (Most of the figures in this book have been generated by MATLAB.)

This chapter introduces MATLAB's high-level 2-D and 3-D plotting facilities. Low-level features, such as Handle Graphics, are discussed later in this chapter.

It should be stressed that the treatment of graphics in this chapter is of necessity brief, and intended to give you a glimpse of the richness and power of MATLAB graphics. For the full treatment, you should consult `help` on the functions mentioned in this chapter, as well as the comprehensive list of graphics functions in Help document which you open up by clicking the "?" and typing graphics in the search tool.

9.1 BASIC 2-D GRAPHS

Graphs (in 2-D) are drawn with the `plot` statement. In its simplest form, it takes a single vector argument as in `plot(y)`. In this case the elements of y are plotted against their indexes, e.g., `plot(rand(1, 20))` plots 20 random numbers against the integers 1–20, joining successive points with straight lines, as in Figure 9.1. If y is a matrix, its columns are plotted against element indexes.

CONTENTS

http://dx.doi.org/10.1016/B978-0-12-394398-9.00009-5

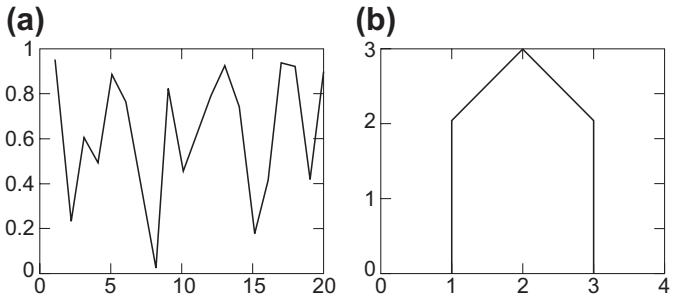

FIGURE 9.1 Examples of `plot`.

Axes are automatically scaled and drawn to include the minimum and maximum data points.

Probably the most common form of `plot` is `plot(x, y)` where `x` and `y` are vectors of the same length, e.g.,

```
x = 0:pi/40:4*pi;
plot(x, sin(x))
```

In this case, the co-ordinates of the ith point are x_i, y_i. This form of `plot` has been used widely in earlier chapters.

Straight-line graphs are drawn by giving the x and y co-ordinates of the end-points in two vectors. For example, to draw a line between the point with cartesian co-ordinates (0, 1) and (4, 3) use the statement:

```
plot([0 4], [1 3])
```

i.e., `[0 4]` contains the x co-ordinates of the two points, and `[1 3]` contains their y co-ordinates.

MATLAB has a set of "easy-to-use" plotting commands, all starting with the string 'ez.' The easy-to-use form of `plot` is `ezplot`, e.g.,

```
ezplot('tan(x)')
```

Exercises

1. Draw lines joining the following points: (0, 1), (4, 3), (2, 0) and (5, –2).
2. See if you can draw a "house" similar to the one depicted in Figure 9.1.

9.1.1 Labels

Graphs may be labeled with the following statements:

```
gtext('text')
```

writes a string (`'text'`) in the graph window. `gtext` puts a cross-hair in the graph window and waits for a mouse button or keyboard key to be pressed. The cross-hair can be positioned with the mouse or the arrow keys. For example:

```
gtext( 'X marks the spot' )
```

Go on—try it!
Text may also be placed on a graph interactively with **Tools–> Edit Plot** from the figure window.

```
grid
```

adds/removes grid lines to/from the current graph. The grid state may be toggled.

`text(x, y, 'text')`

writes text in the graphics window at the point specified by `x` and `y`.

If `x` and `y` are vectors, the text is written at each point. If the text is an indexed list, successive points are labeled with corresponding rows of the text.

`title('text')`

writes the text as a title on top of the graph.

`xlabel('horizontal')`

labels the *x*-axis.

`ylabel('vertical')`

labels the *y*-axis.

9.1.2 Multiple plots on the same axes

There are at least three ways of drawing multiple plots on the same set of axes (which may however be rescaled if the new data falls outside the range of the previous data).

1. The easiest way is simply to use `hold` to keep the current plot on the axes. All subsequent plots are added to the axes until `hold` is released, either with `hold off`, or just `hold`, which toggles the hold state.
2. The second way is to use `plot` with multiple arguments, e.g.,

```
plot(x1, y1, x2, y2, x3, y3, ...)
```

plots the (vector) pairs (`x1`, `y1`), (`x2`, `y2`), and so on. The advantage of this method is that the vector pairs may have different lengths. MATLAB automatically selects a different color for each pair.

If you are plotting two graphs on the same axes you may find plotyy useful—it allows you to have independent y-axis labels on the left and the right, e.g.,

```
plotyy(x,sin(x), x, 10*cos(x))
```

(for x suitably defined).

3. The third way is to use the form:

```
plot(x, y)
```

where x and y may both be matrices, or where one may be a vector and one a matrix.

If one of x or y is a matrix and the other is a vector, the rows or columns of the matrix are plotted against the vector, using a different color for each. Rows or columns of the matrix are selected depending on which have the same number of elements as the vector. If the matrix is square, columns are used.

If x and y are both matrices of the same size, the columns of x are plotted against the columns of y.

If x is not specified, as in `plot(y)`, where y is a matrix, the columns of y are plotted against the row index.

9.1.3 Line styles, markers and color

Line styles, markers and colors may be selected for a graph with a string argument to `plot`, e.g.,

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

joins the plotted points with dashed lines, whereas:

```
plot(x, y, 'o')
```

draws circles at the data points with no lines joining them. You can specify all three properties, e.g.,

```
plot(x,sin(x), x, cos(x), 'om--')
```

plots sin(x) in the default style and color and cos(x) with circles joined with dashes in magenta. The available colors are denoted by the symbols `c`, `m`, `y`, `k`, `r`, `g`, `b`, `w`. You can have fun trying to figure out what they mean, or you can use `help plot` to see the full range of possible symbols.

9.1.4 Axis limits

Whenever you draw a graph with MATLAB it automatically scales the axis limits to fit the data. You can override this with:

```
axis( [xmin, xmax, ymin, ymax] )
```

which sets the scaling on the *current* plot, i.e., draw the graph first, then reset the axis limits.

If you want to specify one of the minimum or maximum of a set of axis limits, but want MATLAB to autoscale the other, use `Inf` or `-Inf` for the autoscaled limit.

You can return to the default of automatic axis scaling with:

```
axis auto
```

The statement:

```
v = axis
```

returns the current axis scaling in the vector `v`.
Scaling is frozen at the current limits with:

```
axis manual
```

so that if `hold` is turned on, subsequent plots will use the same limits.
If you draw a circle, e.g., with the statements:

```
x = 0:pi/40:2*pi;
plot(sin(x), cos(x))
```

the circle probably won't appear round, especially if you resize the figure window. The command:

```
axis equal
```

makes unit increments along the *x*- and *y*-axis the same physical length on the monitor, so that circles always appear round. The effect is undone with `axis normal`.

You can turn axis labeling and tick marks off with `axis off`, and back on again with `axis on`.

```
axes and axis?
```

You and I might be forgiven for thinking that the word "axes" denotes the plural of "axis," which it does indeed in common English usage. However, in MATLAB the word "axes" refers to a particular graphics *object*, which includes not only the *x*-axis and *y*-axis and their tick marks and labels, but also everything drawn on those particular axes: the actual graphs and any text included in the figure. Axes objects are discussed in more detail later in this chapter.

9.1.5 Multiple plots in a figure: `subplot`

You can show a number of plots in the same figure window with the `subplot` function. It looks a little curious at first, but it's quite easy to get the hang of it. The statement:

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

divides the figure window into $m \times n$ small sets of axes, and selects the pth set for the current plot (numbered by row from the left of the top row). For example, the following statements produce the four plots shown in Figure 9.2 (details of 3-D plotting are discussed in Section 9.2).

```
[x, y] = meshgrid(-3:0.3:3);
z = x .* exp(-x.^2 - y.^2);
subplot(2,2,1)
mesh(z),title('subplot(2,2,1)')
subplot(2,2,2)
mesh(z)
view(-37.5,70),title('subplot(2,2,2)')
subplot(2,2,3)
mesh(z)
view(37.5,-10),title('subplot(2,2,3)')
subplot(2,2,4)
mesh(z)
view(0,0),title('subplot(2,2,4)')
```

The command `subplot(1,1,1)` goes back to a single set of axes in the figure.

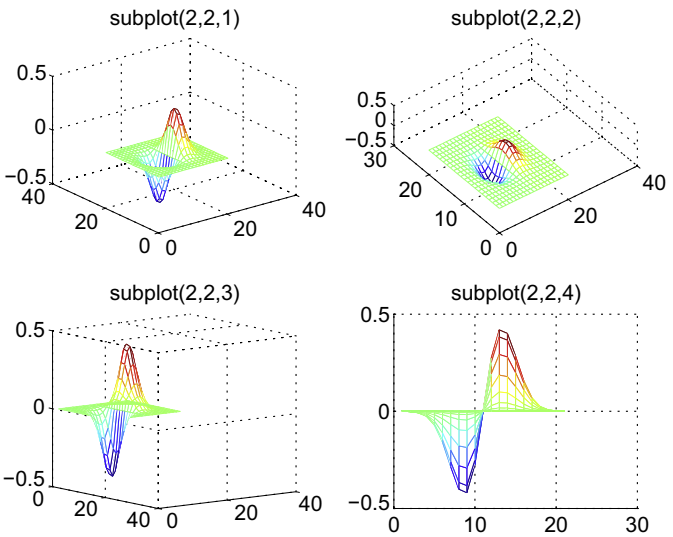

FIGURE 9.2 Four subplots: rotations of a 3-D surface.

9.1.6 `figure`, `clf` and `cla`

`figure(h)`, where `h` is an integer, creates a new figure window, or makes figure `h` the current figure. Subsequent plots are drawn in the current figure. `h` is called the *handle* of the figure. Handle Graphics is discussed further in a later section of this chapter.

`clf` clears the current figure window. It also resets all properties associated with the axes, such as the hold state and the axis state.

`cla` deletes all plots and text from the current axes, i.e., leaves only the x- and y-axes and their associated information.

9.1.7 Graphical input

The command:

```
[x, y] = ginput
```

allows you to select an unlimited number of points from the current graph using a mouse or arrow keys. A movable cross-hair appears on the graph. Clicking saves its co-ordinates in `x(i)` and `y(i)`. Pressing **Enter** terminates the input. An example is provided in the last chapter of this book which involves selecting points in a figure and fitting a curve to them.

The command:

```
[x, y] = ginput(n)
```

works like `ginput` except that you must select exactly `n` points.

See `help` for further information.

9.1.8 Logarithmic plots

The command:

```
semilogy(x, y)
```

plots y with a $\log_{10}$ scale and x with a linear scale. For example, the statements:

```
x = 0:0.01:4;
semilogy(x, exp(x)), grid
```

produce the graph in Figure 9.3. Equal increments along the y-axis represent multiples of powers of 10. So, starting from the bottom, the grid lines are drawn at 1, 2, 3, …, 10, 20, 30 …, 100, … . Incidentally, the graph of e^x on these axes is a straight line, because the equation $y = e^x$ transforms into a linear equation when you take logs of both sides.

See also `semilogx` and `loglog`.

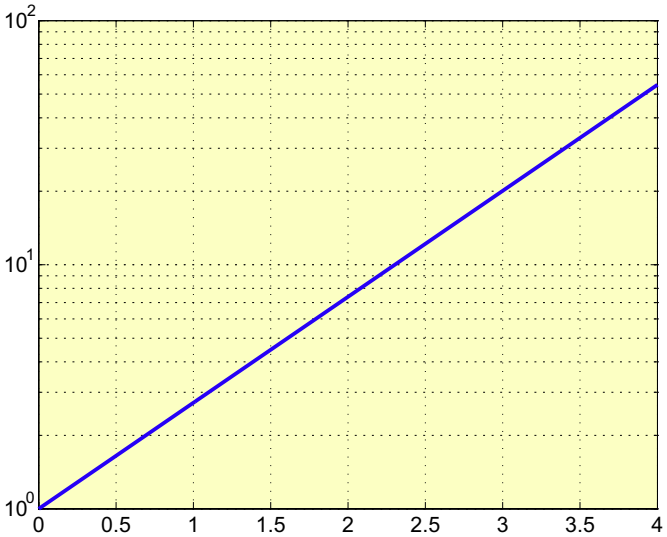

FIGURE 9.3 A logarithmic plot.

Note that `x` and `y` may be vectors and/or matrices, just as in `plot`.

Exercise

Draw graphs of x^2, x^3, x^4 and e^{x^2} over the interval $0 \leqslant x \leqslant 4$, using semilogy.

9.1.9 Polar plots

The point $(x,\ y)$ in cartesian co-ordinates is represented by the point $(\theta,\ r)$ in *polar* co-ordinates, where:

$$x = r\cos(\theta),$$
$$y = r\sin(\theta),$$

and θ varies between 0 and 2π radians $(360°)$.

The command:

```
polar(theta, r)
```

generates a polar plot of the points with angles in theta and magnitudes in `r`.

As an example, the statements:

```
x = 0:pi/40:2*pi;
polar(x, sin(2*x)),grid
```

produce the plot shown in Figure 9.4.

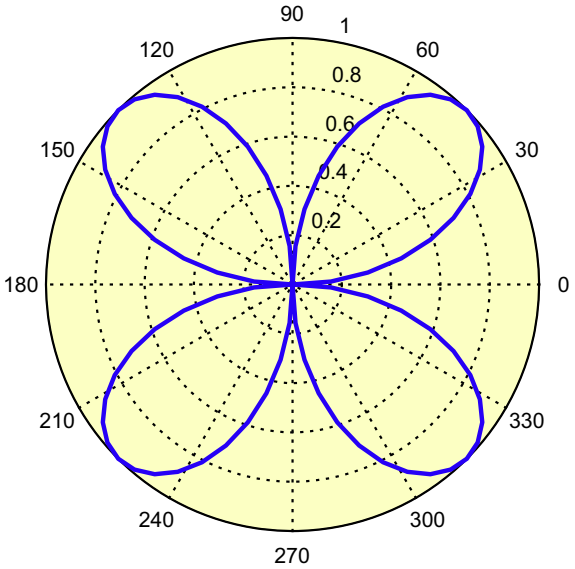

FIGURE 9.4 Polar plot of θ against $\sin(2\theta)$.

9.1.10 Plotting rapidly changing mathematical functions: `fplot`

In all the graphing examples so far, the x co-ordinates of the points plotted have been incremented uniformly, e.g., `x = 0:0.01:4`. If the function being plotted changes very rapidly in some places, this can be inefficient, and can even give a misleading graph.

For example, the statements:

```
x = 0.01:0.001:0.1;
plot(x, sin(1./x))
```

produce the graph shown in Figure 9.5a. But if the x increments are reduced to 0.0001, we get the graph in Figure 9.5b instead. For x<0.04, the two graphs look quite different.

MATLAB has a function called `fplot` which uses a more elegant approach. Whereas the above method evaluates $\sin(1/x)$ at equally spaced intervals, `fplot` evaluates it more frequently over regions where it changes more rapidly. Here's how to use it:

```
fplot('sin(1/x)', [0.01 0.1]) % no, 1./x not needed!
```

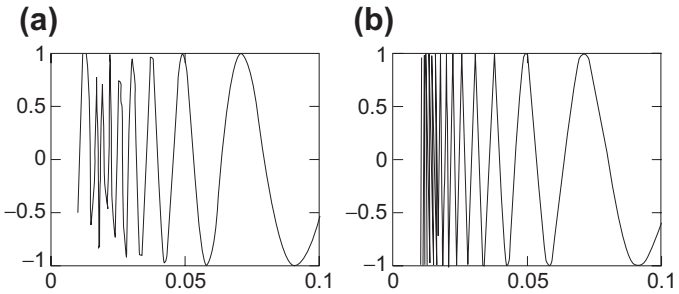

FIGURE 9.5 $y = \sin(1/x)$.

9.1.11 The Property Editor

The most general way of editing a graph is by using the Property Editor, e.g., **Edit–> Figure Properties** from the figure window. This topic is discussed briefly towards the end of this chapter.

9.2 3-D PLOTS

MATLAB has a variety of functions for displaying and visualizing data in 3-D, either as lines in 3-D, or as various types of surfaces. This section provides a brief overview.

9.2.1 `plot3`

The function plot3 is the 3-D version of plot. The command:

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

draws a 2-D projection of a line in 3-D through the points whose co-ordinates are the elements of the vectors x, y and z. For example, the command:

```
plot3(rand(1,10), rand(1,10), rand(1,10))
```

generates 10 random points in 3-D space, and joins them with lines, as shown in Figure 9.6a.

As another example, the statements:

```
t = 0:pi/50:10*pi;
plot3(exp(-0.02*t).*sin(t), exp(-0.02*t).*cos(t),t), ...
xlabel('x-axis'), ylabel('y-axis'), zlabel('z-axis')
```

produce the inwardly spiraling helix shown in Figure 9.6b. Note the orientation of the *x*-, *y*- and *z*-axes, and in particular that the *z*-axis may be labeled with `zlabel`.

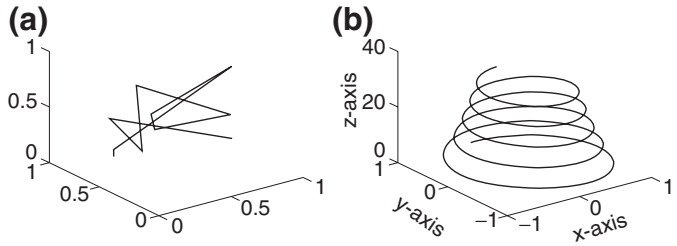

FIGURE 9.6 Examples of `plot3`.

9.2.2 Animated 3-D plots with `comet3`

The function `comet3` is similar to `plot3` except that it draws with a moving "comet head." Use `comet3` to animate the helix in Figure 9.6b. No prizes for guessing the name of the 2-D version.

9.2.3 Mesh surfaces

In Chapter 1 we saw how to draw the Mexican hat (Figure 1.13):

```
[x y] = meshgrid(-8 : 0.5 : 8);

r = sqrt(x.^2 + y.^2) + eps;

z = sin(r) ./ r;
```

This drawing is an example of a *mesh surface*.

To see how such surface is drawn, let's take a simpler example, say $z = x^2 - y^2$. The surface we are after is the one generated by the values of z as we move around the x-y plane. Let's restrict ourselves to part of the first quadrant of this plane, given by:

$$0 \leqslant x \leqslant 5, \quad 0 \leqslant y \leqslant 5.$$

The first step is to set up the *grid* in the x-y plane over which the surface is to be plotted. You can use the MATLAB function `meshgrid` to do it, as follows:

```
[x y] = meshgrid(0:5);
```

This statement sets up two matrices, `x` and `y`. (Functions, such as `meshgrid`, which return more than one "output argument," are discussed in detail in Chapter 10. However, you don't need to know the details in order to be able to use it here.)

The two matrices in this example are:

```
x =
     0     1     2     3     4     5
     0     1     2     3     4     5
     0     1     2     3     4     5
     0     1     2     3     4     5
     0     1     2     3     4     5
     0     1     2     3     4     5

y =
     0     0     0     0     0     0
     1     1     1     1     1     1
     2     2     2     2     2     2
     3     3     3     3     3     3
     4     4     4     4     4     4
     5     5     5     5     5     5
```

The effect of this is that the *columns* of the matrix `x` as it is displayed hold the x co-ordinates of the points in the grid, while the *rows* of the display of `y` hold the y co-ordinates. Recalling the way MATLAB array operations are defined, element by element, this means that the statement:

```
z = x.^2 - y.^2
```

will correctly generate the surface points:

```
z =
     0     1     4     9    16    25
    -1     0     3     8    15    24
    -4    -3     0     5    12    21
    -9    -8    -5     0     7    16
   -16   -15   -12    -7     0     9
   -25   -24   -21   -16    -9     0
```

For example, at the grid point $(5,\ 2)$, z has the value $5^2 - 2^2 = 21$. Incidentally, you don't need to worry about the exact relationship between grid co-ordinates and matrix subscripts; this is taken care of by `meshgrid`.

The statement `mesh(z)` then plots the surface (Figure 9.7), with mesh lines connecting the points in the surface that lie above grid points.

Note that `mesh(z)` shows the row and column indices (subscripts) of the matrix `z` on the x and y axes. If you want to see proper values on the x and y axes use `mesh(x,y,z)`. This applies to many of the other 3-D plotting functions.

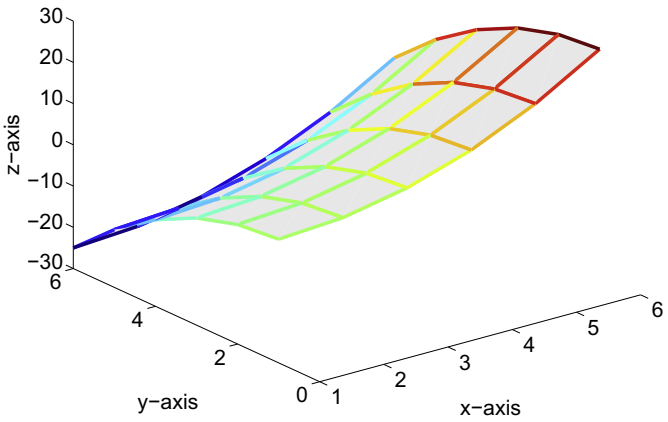

FIGURE 9.7 The surface $z = x^2 - y^2$.

The function `mesh` draws a surface as a "wire frame." An alternative visualization is provided by `surf`, which generates a faceted view of the surface (in color), i.e., the wire frame is covered with small tiles.

See `help` for variations on `mesh` and `surf`.

Exercises

1. Draw the surface shown in Figure 9.7 with a finer mesh (of 0.25 units in each direction), using:

```
[x y] = meshgrid(0:0.25:5);
```

(the number of mesh points in each direction is 21).

2. The initial heat distribution over a steel plate is given by the function:

$$u(x, y) = 80y^2 e^{-x^2 - 0.3y^2}.$$

Plot the surface u over the grid defined by:

$$-2.1 \leqslant x \leqslant 2.1, \quad -6 \leqslant y \leqslant 6,$$

where the grid width is 0.15 in both directions. You should get the plot shown in Figure 9.8.

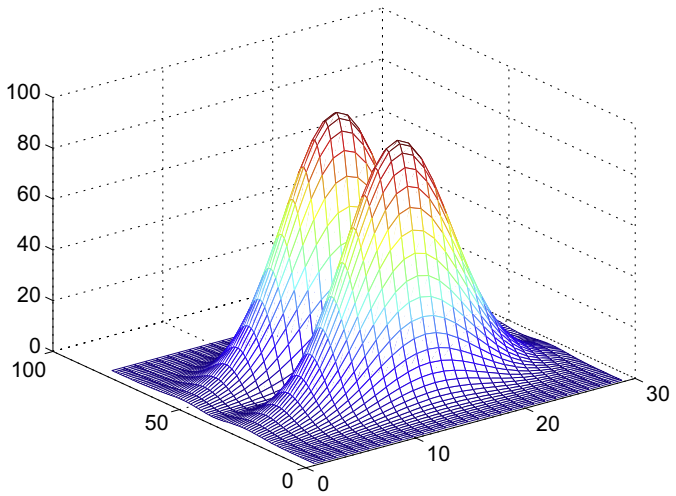

FIGURE 9.8 Heat distribution over a steel plate.

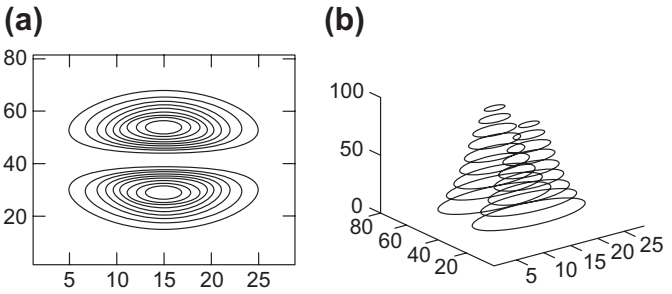

FIGURE 9.9 Contour plots.

9.2.4 Contour plots

If you managed to draw the plot in Figure 9.8, try the command:

```
contour(u)
```

You should get a *contour plot* of the heat distribution, as shown in Figure 9.9a, i.e., the *isothermals* (lines of equal temperature). Here's the code:

```
[x y] = meshgrid(-2.1:0.15:2.1, -6:0.15:6); % x- y-grids di
u = 80 * y.^2 .* exp(-x.^2 - 0.3*y.^2);
contour(u)
```

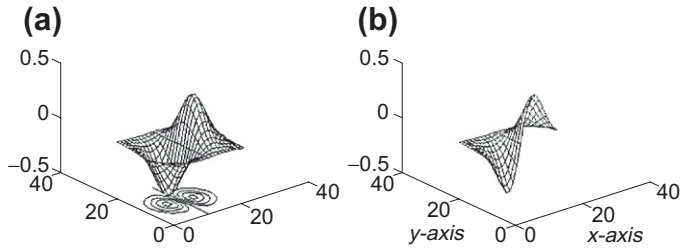

FIGURE 9.10 (a) `meshc`; (b) cropping a surface.

The function `contour` can take a second input variable. It can be a scalar specifying how many contour levels to plot, or it can be a vector specifying the values at which to plot the contour levels.

You can get a 3-D contour plot with `contour3`, as shown in Figure 9.9b.

Contour levels may be labeled with `clabel` (see `help`).

A 3-D contour plot may be drawn under a surface with `meshc` or `surfc`. For example, the statements:

```
[x y] = meshgrid(-2:.2:2);
z = x .* exp(-x.^2 - y.^2);
meshc(z)
```

produce the graph in Figure 9.10a.

9.2.5 Cropping a surface with `NaN`s

If a matrix for a surface plot contains `NaN`s, these elements are not plotted. This enables you to cut away (crop) parts of a surface. For example, the statements:

```
[x y] = meshgrid(-2:.2:2, -2:.2:2);
z = x .* exp(-x.^2 - y.^2);
c = z;        % preserve the original surface
c(1:11,1:21) = nan*c(1:11,1:21);
mesh(c), xlabel('x-axis'), ylabel('y-axis')
```

produce the graph in Figure 9.10b.

9.2.6 Visualizing vector fields

The function `quiver` draws little arrows to indicate a gradient or other vector field. Although it produces a 2-D plot, it's often used in conjunction with `contour`, which is why it's described briefly here.

As an example, consider the scalar function of two variables $V = x^2 + y$. The *gradient* of V is defined as the *vector* field:

$$\nabla V = \left(\frac{\partial V}{\partial x}, \frac{\partial V}{\partial y} \right)$$
$$= (2x, 1).$$

The following statements draw arrows indicating the direction of ∇V at points in the x-y plane (see Figure 9.11):

```
[x y] = meshgrid(-2:.2:2, -2:.2:2);
V = x.^2 + y;
dx = 2*x;
dy = dx;            % dy same size as dx
dy(:,:) = 1;        % now dy is same size as dx but all 1's
contour(x, y, V), hold on
quiver(x, y, dx, dy), hold off
```

The "contour" lines indicate families of *level surfaces*; the gradient at any point is perpendicular to the level surface which passes through that point. The vectors `x` and `y` are needed in the call to `contour` to specify the axes for the contour plot.

An additional optional argument for `quiver` specifies the length of the arrows. See `help`.

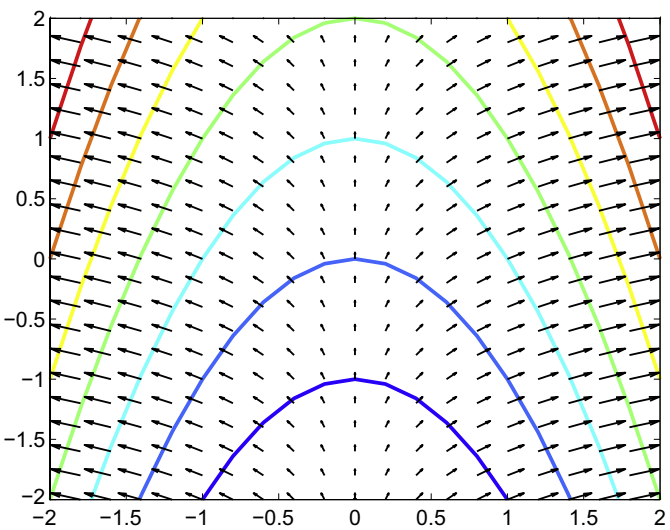

FIGURE 9.11 Gradients and level surfaces.

If you can't (or don't want to) differentiate V, you can use the `gradient` function to estimate the derivative:

```
[dx dy] = gradient(V, 0.2, 0.2);
```

The values 0.2 are the increments in the x and y directions used in the approximation.

9.2.7 Visualization of matrices

The `mesh` function can also be used to "visualize" a matrix. The following statements generate the plot in Figure 9.12:

```
a = zeros(30,30);
a(:,15) = 0.2*ones(30,1);
a(7,:) = 0.1*ones(1,30);
a(15,15) = 1;
mesh(a)
```

The matrix `a` is 30×30. The element in the middle—`a(15,15)`—is 1, all the elements in row 7 are 0.1, and all the remaining elements in column 15 are 0.2. `mesh(a)` then interprets the rows and columns of `a` as an x-y co-ordinate grid, with the values `a(i,j)` forming the mesh surface above the points (`i`, `j`).

The function `spy` is useful for visualizing sparse matrices.

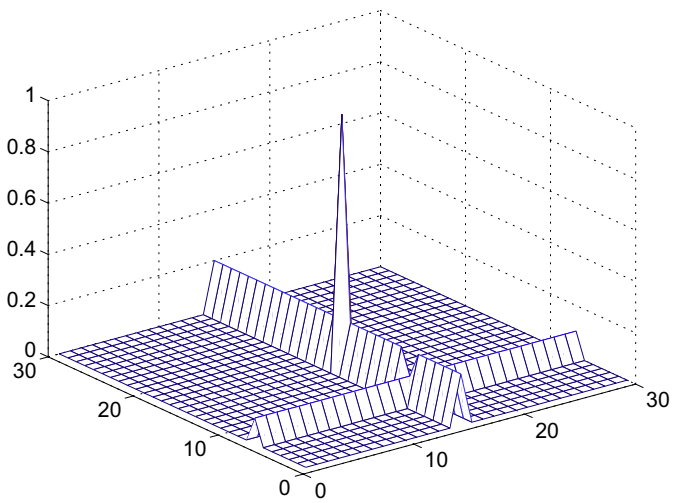

FIGURE 9.12 Visualization of a matrix.

9.2.8 Rotation of 3-D graphs

The `view` function enables you to specify the angle from which you view a 3-D graph. To see it in operation, run the following program, which rotates the visualized matrix in Figure 9.12:

```
a = zeros(30,30);
a(:,15) = 0.2*ones(30,1);
a(7,:) = 0.1*ones(1,30);
a(15,15) = 1;
el = 30;
for az = -37.5:15:-37.5+360
  mesh(a), view(az, el)
  pause(0.5)
end
```

The function `view` takes two arguments. The first one, `az` in this example, is called the *azimuth* or polar angle in the *x*-*y* plane (in degrees). `az` rotates the *viewpoint* (you) about the *z*-axis—i.e., about the "pinnacle" at (15, 15) in Figure 9.12—in a counter-clockwise direction. The default value of `az` is −37.5°. The program therefore rotates you in a counter-clockwise direction about the *z*-axis in 15° steps starting at the default position.

The second argument of `view` is the vertical elevation `el` (in degrees). This is the angle a line from the viewpoint makes with the *x*-*y* plane. A value of 90° for `el` means you are directly overhead. Positive values of the elevation mean you are above the *x*-*y* plane; negative values mean you are below it. The default value of `el` is 30°.

The command `pause(n)` suspends execution for `n` seconds.

You can rotate a 3-D figure interactively as follows. Click the Rotate 3-D button in the figure toolbar (first button from the right). Click on the axes and an outline of the figure appears to help you visualize the rotation. Drag the mouse in the direction you want to rotate. When you release the mouse button the rotated figure is redrawn.

Exercise

Rewrite the above program to change the elevation gradually, keeping the azimuth fixed at its default.

9.3 HANDLE GRAPHICS

The richness and power of MATLAB graphics is made possible by its *Handle Graphics objects*. There is a very helpful section devoted to Handle Graphics in the online documentation **MATLAB Help: Graphics**. What follows is of necessity a brief summary of its main features.

Handle Graphics objects are the basic elements used in MATLAB graphics. The objects are arranged in a parent-child inheritance structure as shown in Figure 9.13. For example, Line and Text objects are children of Axes objects. This is probably the most common parent-child relationship. Axes objects define a region in a figure window and orient their children within this region. The actual plots in an Axes object are Line objects. The Axis labels and any text annotations are Text objects. It is important to be aware of this parent-child hierarchy when you want to manipulate graphics objects using their handles.

Well, what exactly is the handle of a graphics object? Whenever MATLAB creates a graphics objects it automatically creates a handle to that object. You can get the handle of the object using a function that explicitly returns the handle of a graphics object, or you can create the handle explicitly when you draw the graphics object. The handle itself has a floating point representation, but its actual value need not concern you. What is more important is to save its name and then to use the handle to change or manipulate your graphics object.

The root object is the only object whose handle is 0. There is only one root object, created by MATLAB at startup. All other objects are its descendants, as you can see in Figure 9.13.

9.3.1 Getting handles

Here's how to get handles:

- The functions that draw graphics objects can also be used to return the handle of the object drawn, e.g.,

```
x = 0:pi/20:2*pi;
hsin = plot(x, sin(x))
hold on
hx = xlabel('x')
```

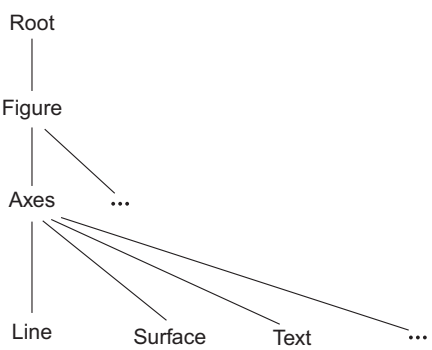

FIGURE 9.13 Parent-child relationships of Handle Graphics objects (from top to bottom).

`hsin` is the handle of the Line object (the sine graph), and `hx` is the handle of the Text object (the *x* axis label) in the current Axes object.

The command:

```
figure(h)
```

where `h` is an integer, creates a new figure, or makes figure `h` the current figure. `h` is the handle of the figure object.

- There are three functions that return the handle of particular graphics objects:

  `gcf` gets the handle of the current figure, e.g.,

  ```
  hf = gcf;
  ```

  `gca` gets the handle of the current axes.

  `gco` gets the handle of the current graphics object, which is the last graphics object created or clicked on. For example, draw the sine graph above and get its handle `hsin`. Click on the graph in the figure window. Then enter the command:

  ```
  ho = gco
  ```

in the Command Window. `ho` should be set to the handle of the sine graph (it should have the same numeric value as `hsin`).

9.3.2 Graphics object properties and how to change them

Once you have the handle of a graphics object you can use it to change the properties of that object. As an example, draw a sine graph and get its handle, as demonstrated above:

```
x = 0:pi/20:2*pi;
hsin = plot(x, sin(x))
```

Now suppose you want to make the graph much thicker. Enter the following command:

```
set(hsin, 'linewidth', 4);
```

You should get a nice fat sine curve!

`linewidth` is just one of the many properties of our graphics object. To see all the property names of an object and their current values use `get(h)` where `h` is the object's handle. In the case of our sine graph:

```
get(hsin)
 Color = [0 0 1]
 EraseMode = normal
 LineStyle = -
 LineWidth = [4]
 Marker = none
 MarkerSize = [6]
 MarkerEdgeColor = auto
 MarkerFaceColor = none
 XData = [ (1 by 41) double array]
 YData = [ (1 by 41) double array]
 ZData = []

 BeingDeleted = off
 ButtonDownFcn =
 Children = []
 Clipping = on
 CreateFcn =
 DeleteFcn =
 BusyAction = queue
 HandleVisibility = on
 HitTest = on
 Interruptible = on
 Parent = [100.001]
 Selected = off
 SelectionHighlight = on
 Tag =
 Type = line
 UIContextMenu = []
 UserData = []
 Visible = on
```

You can change any property value with the `set` function:

`set(`*handle, 'PropertyName', PropertyValue*`)`

The command `set(`*handle*`)` lists all the possible property values (where appropriate).

You can get an object's handle and change its properties all in the same breath. For example:

```
set(gcf, 'visible', 'off')
```

makes the current figure invisible (without closing it—i.e., it's still "there"). No prizes for guessing how to make it visible again … .

Property names are not case-sensitive, and you can abbreviate them to a few letters to make them unique. For example, you can abbreviate the `type` property to `ty`:

```
get(hsin,'ty')
ans =
line
```

(helpful if you don't know what type of objects you are dealing with!).

The different types of graphics objects don't all have the same set of properties, although there are some properties which are common to all graphics objects, such as `children`, `parent`, `type`, and so on.

9.3.3 A vector of handles

If a graphics object has a number of children the `get` command used with the `children` property returns a vector of the children's handles. Sorting out the handles is then quite fun, and demonstrates why you need to be aware of the parent-child relationships!

As an example, plot a continuous sine graph and an exponentially decaying sine graph marked with `o`'s in the same figure:

```
x = 0:pi/20:4*pi;
plot(x, sin(x))
hold on
plot(x, exp(-0.1*x).*sin(x), 'o')
hold off
```

Now enter the command:

```
hkids = get(gca,'child')
```

You will see that a vector of handles with two elements is returned. The question is, which handle belongs to which plot? The answer is that the handles of children of the axes are returned in the *reverse order in which they are created*, i.e., `hkids(1)` is the handle of the exponentially decaying graph, while `hkids(2)` is the handle of the sine graph. So now let's change the markers on the decaying graph, and make the sine graph much bolder:

```
set(hkids(1), 'marker', '*')
set(hkids(2), 'linew', 4)
```

You should get the plots shown in Figure 9.14.

If you are desperate and don't know the handles of any of your graphics objects you can use the `findobj` function to get the handle of an object with a property value that uniquely identifies it. In the original version of the plots in Figure 9.14, the decaying plot can be identified by its `marker` property:

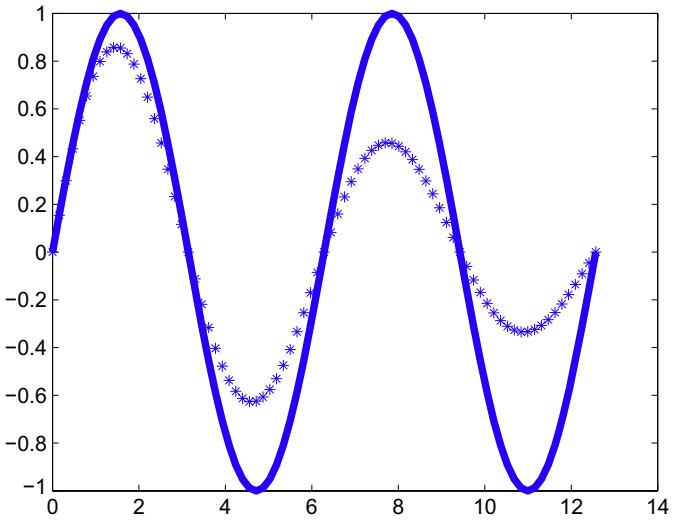

FIGURE 9.14 Result of manipulating a figure using the handles of axes children.

```
hdecay = findobj('marker', 'o' )
```

9.3.4 Graphics object creation functions

Each of the graphics objects shown in Figure 9.13 (except the root object) has a corresponding creation function, named after the object it creates. See `help` for details.

9.3.5 Parenting

By default, all graphics objects are created in the current figure. However, you can specify the parent of an object when you create it. For example:

```
axes('Parent',figure_handle, ... )
```

creates an axes in the figure with handle *figure_handle*. You can also move an object from one parent to another by redefining its `parent` property:

```
set(gca, 'Parent',figure_handle)
```

9.3.6 Positioning figures

At startup MATLAB determines the default position and size of the figure window, according to the size of your computer screen. You can change this by setting the `Position` property of the figure object.

Before you tamper with the figure's `Position` property you need to know the dimensions of your screen—this is one of the root object's properties. For example:

```
get(0, 'screensize')
ans =
     1     1   800   600
```

i.e., my screen is 800 by 600 pixels. The units of `Screensize` are pixels by defaults. You can (naturally) change the units by setting the root's `Units` property. For example:

```
set(0, 'units', 'normalized')
```

normalizes the width and height of the screen to 1. This is useful when you are writing an M-file which is to be run on different computer systems.

Having sorted out the size of your screen, you can then fiddle with the figure `Position` property, which is defined as a vector:

```
[left bottom width height]
```

`left` and `bottom` define the position of the first addressable pixel in the lower left corner of the window, specified with respect to the lower left corner of the screen. `width` and `height` define the size of the interior of the window (excluding the window border).

You can normalize the figure's `Unit` property as well as the root's `Unit` property. You can then position figures absolutely, without reference to variations in screen size. For example, the following code normalizes units and creates two figures in the upper half of the screen:

```
set(0, 'units', 'normalized')
h1 = figure('units', 'normalized', 'visible', 'off')
h2 = figure('units', 'normalized', 'visible', 'off')
set(h1, 'position', [0.05 0.5 0.45 0.35], 'visible', 'on')
set(h2, 'position', [0.53 0.5 0.45 0.35], 'visible', 'on')
```

Note that the `Visible` property is first set to `off` to avoid the figures being drawn in the default position when they are created. They are only drawn when their positions have been redefined. Alternatively you can do everything when the figures are created:

```
h1 = figure('un', 'normalized', 'pos',
[0.05 0.5 0.45 0.35])
```

9.4 EDITING PLOTS

There are a number of ways of editing plots.

9.4.1 Plot edit mode

To see how this works draw a graph, e.g., the friendly old sine. There are several ways to activate plot edit mode:

- Select **Tools -> Edit Plot** in the figure window.
- Click on the Edit Plot selection button in the figure window toolbar (arrow pointing roughly north-west).
- Run the `plotedit` command in the Command Window.When a figure is in plot edit mode the toolbar selection button is highlighted. Once you are in plot edit mode select an object by clicking on it. Selection handles will appear on the selected object.

As an exercise, get the sine graph into plot edit mode and try the following:

- Select the graph (click on it). Selection handles should appear.
- Right click on the selected object (the graph). A context menu appears.
- Use the context menu to change the graph's line style and color.
- Use the **Insert** menu to insert a legend (although this makes more sense where you have multiple plots in the figure).
- Insert a text box inside the figure close to the graph as follows. Click on the Insert Text selection button in the toolbar (the capital A). The cursor changes shape to indicate that it is in text insertion mode. Move the insertion point until it touches the graph somewhere and click. A text box appears. Enter some text in it.
  You can use a subset of TEX characters and commands in text objects. For example, the text x_k in Figure 14.1 was produced with the string `{\itx_k`. See the entry `Text property` under `String[1]` in the online Help for a list of a available TEX characters and commands.
- Having labeled the graph you can change the format of the labels. Select the label and right-click. Change the font size and font style.
- Play around with the Insert Arrow and Insert Line selection buttons on the toolbar to see if you can insert lines and arrows on the graph.

To exit plot edit mode, click the selection button or uncheck the **Edit Plot** option on the **Tools** menu.

9.4.2 Property Editor

The Property Editor is more general than plot edit mode. It enables you to change object properties interactively, rather than with the `set` function. It is ideal for preparing presentation graphics.

There are numerous ways of starting the Property Editor (you may already have stumbled onto some):

- If plot edit mode is enabled you can:
  - Double-click on an object.
  - Right-click on an object and select **Properties** from the context menu.
- Select **Figure Properties**, **Axes Properties** or **Current Object Properties** from the figure **Edit** menu.
- Run the `propedit` command on the command line.

To experiment with the Property Editor it will be useful to have multiple plots in a figure:

```
x = 0:pi/20:2*pi;
hsin = plot(x,sin(x))
hold on
hcos = plot(x,cos(x))
hold off
```

Start the Property Editor and work through the following exercises:

- The navigation bar at the top of the Property Editor (labeled **Edit Properties for:**) identifies the object being edited. Click on the down arrow at the right of the navigation bar to see all the objects in the figure. You will notice that there are two line objects. Immediately we are faced with the problem of identifying the two line objects in our figure. The answer is to give each of them tags by setting their `Tag` properties.

  Go back to the figure and select the sine graph. Back in the Property Editor the navigation bar indicates you are editing one of the line objects. You will see three tabs below the navigation bar: **Data**, **Style** and **Info**. Click the **Info** tab, and enter a label in the **Tag** box, e.g., `sine`. Press **Enter**. The tag `sine` immediately appears next to the selected line object in the navigation bar.

  Give the cosine graph a tag as well (start by selecting the other line object).
- Select the sine graph. This time select the **Style** tab and go beserk changing its color, line style, line width and markers.
- Now select the axes object. Use the **Labels** tab to insert some axis labels. Use the **Scale** tab to change the y axis limits, for example.

  Note that if you were editing a 3-D plot you would be able to use the **Viewpoint** tab to change the viewing angle and to set various camera properties.

Have fun!

9.5 ANIMATION

There are three facilities for animation in MATLAB:

- The `comet` and `comet3` functions can be used to draw comet plots.
- The `getframe` function may be used to generate "movie frames" from a sequence of graphs. The `movie` function can then be used to play back the movie a specified number of times.
  The MATLAB online documention has the following script in **MATLAB Help: Graphics: Creating Specialized Plots: Animation**. It generates 16 frames from the Fast Fourier Transforms of complex matrices:

```
for k = 1:16
    plot(fft(eye(k+16)))
    axis equal
    M(k) = getframe;
end
```

Now play it back, say five times:

```
movie(M, 5)
```

You can specify the speed of the playback, among other things. See `help`.

- The most versatile (and satisfying) way of creating animations is by using the Handle Graphics facilities. Two examples follow.

9.5.1 Animation with Handle Graphics

For starters, run the following script, which should show the marker `o` tracing out a sine curve, leaving a trail behind it:

```
% animated sine graph
x = 0;
y = 0;
dx = pi/40;
p = plot(x, y, 'o', 'EraseMode', 'none'); % 'xor' shows only current point
                                          % ' none' shows all points
axis([0 20*pi -2 2])

for x = dx:dx:20*pi;
    x = x + dx;
    y = sin(x);
    set(p, 'XData', x, 'YData', y)
    drawnow
end
```

Note:

- The statement:

```
p = plot(x, y, 'o', 'EraseMode', 'none');
```

achieves a number of things. It plots the first point of the graph. It saves the handle `p` of the plot for further reference. And it specifies that the `Erasemode` property is `none`, i.e., the object must not be erased when it is drawn again. To achieve complete animation, set this property to `xor`—try it now. Then the object is erased each time it is redrawn (in a slightly different position), creating the classic animation effect.

- The statement:

```
set(p, 'XData', x, 'YData', y)
```

sets the `x` and `y` data values of the object `p` to the new values generated in the `for` loop, and "redraws" the object. However, it is not drawn on the screen immediately—it joins the "event queue," where it waits until it is flushed out.

- Finally, the `drawnow` function flushes the event queue and draws the object on the screen so that we can see the fruit of our labors.

  As `help drawnow` will reveal, there are four events that flush the event queue:

  - a return to the MATLAB prompt—this is how you have seen all the graphs you've drawn up to now;
  - hitting a `pause` statement;
  - executing a `getframe` command;
  - executing a `drawnow` command.

For example, you could make the marker move in a more stately fashion by replacing `drawnow` with `pause(0.05)`—that's 0.05 s.

The next example is based on one in the **Animation** section of the MATLAB documentation. It involves *chaotic* motion described by a system of three non-linear differential equations having a "strange attractor" (known as the Lorenz strange attractor). The system can be written as:

$$\frac{dy}{dt} = \mathbf{A}y,$$

where $y(t)$ is a vector with three components, and $\mathbf{A}$ is a matrix depending on y:

$$\mathbf{A}(y) = \begin{bmatrix} -8/3 & 0 & y(2) \\ 0 & -10 & 10 \\ -y(2) & 28 & -1 \end{bmatrix}.$$

The following script solves the system approximately using Euler's method, and shows the solution orbiting about two different attractors without settling into a steady orbit about either. Figure 9.15 shows the situation after a few thousand points have been plotted.

```
A = [ -8/3 0 0; 0 -10 10; 0 28 -1 ];
y = [35 -10 -7]';
h = 0.01;
p = plot3(y(1), y(2), y(3), 'o', ...
    'erasemode', 'none', 'markersize', 2);
axis([0 50 -25 25 -25 25])
hold on
i = 1;

while 1
    A(1,3) = y(2);
    A(3,1) = -y(2);
    ydot = A*y;
    y = y + h*ydot;
    % Change colour occasionally
    if rem(i,500) == 0
        set(p, 'color', [rand, rand, rand])
    end
    % Change co-ordinates
    set(p, 'XData', y(1), 'YData', y(2), 'ZData', y(3))
    drawnow
    i=i+1;
end
```

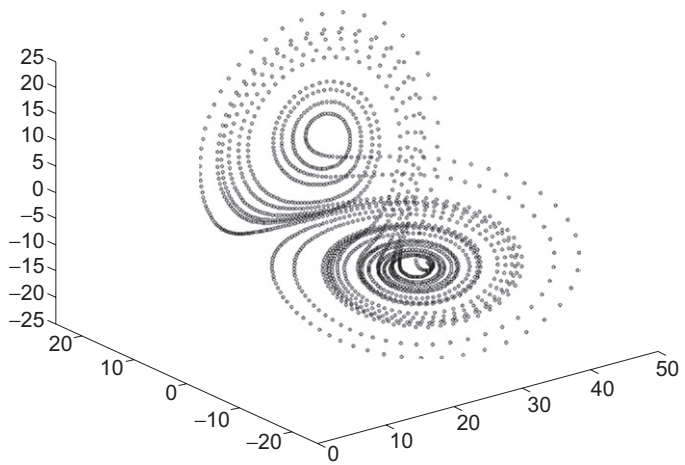

FIGURE 9.15 The Lorenz strange attractor.

If all the points are all plotted in the same color you eventually wouldn't be able to see the new points generated: a large area of the screen would be filled with the drawing color. The color is therefore set randomly after every 500 points are plotted.

9.6 COLOR etc.

9.6.1 Colormaps

MATLAB graphics can generate a rich variety of colors. You may be wondering how.

The following script shows a view of the Earth from space:

```
load earth
image(X); colormap(map)
axis image
```

(`axis image` is the same as `axis equal` except that the plot box fits tightly around the data). For a variation use `hot` as the argument of `colormap` instead of `map`.

The matrix `X` loaded from `earth` is 257-by-250. Each of the elements of `X` is an integer in the range 1–64. For example, here is a 3-by-3 submatrix of `X` (somewhere in north-east Africa):

```
X(39:41,100:102)
ans =
    14    15    14
    10    16    10
    10    10    10
```

The `colormap` function generates by default a 64-by-3 matrix with elements in the range 0–1. The values in the three columns represent the intensities of the red, green and blue (RGB) video components respectively. Each row of this matrix therefore defines a particular color by specifying its RGB components. The `image` function maps each element of its argument to a row in the colormap to find the color of that element. For example, `X(40,101)` has the value 16. Row 16 of the colormap has the three values

```
0.6784 0.3216 0.1922
```

`(reddish)`as you can easily verify with the statements:

```
cm = colormap(map);
cm(16,:)
```

(The `map` colormap is also loaded from `earth`.) These values of RGB specify the color of the pixel 40 from the top and 101 from the left in the figure. Incidentally, you can use the statement:

```
[xp yp] = ginput
```

to get the co-ordinates of a point in the figure (crosshairs appears in the figure; click on the point whose co-ordinates you want). Clicking on the point in north-east Africa results in:

```
xp =
  101.8289  % from the left (column of X)
yp =
   40.7032  % from the top (row of X)
```

Note that `xp` and `yp` correspond to the *columns* and *rows* of `X` respectively (imagine the matrix X superimposed on the figure).

There are a number of functions provided by MATLAB which generate colormaps, e.g., `jet` (the default), `bone`, `flag`, `prism`. See `help graph3d` for a complete list.

You can sample the various colormaps quite nicely with the following statement, which shows 64 vertical strips, each in a different color:

```
image(1:64),colormap(prism)
```

Or why not generate some random colors?

```
randmap(:,1) = rand(64,1);
randmap(:,2) = rand(64,1);
randmap(:,3) = rand(64,1);
image(1:64);colormap(randmap)
```

The function `colorbar` displays the current colormap vertically or horizontally in the figure with your graph, indicating how the 64 colors are mapped. Try it with the image of the Earth.

Note that 64 is the default length of a colormap. The functions that generate colormaps have an optional parameter specifying the length of the colormap.

9.6.2 Color of surface plots

When you draw a surface plot with a single matrix argument, e.g., `surf(z)`, the argument `z` specifies both the height of the surface and the color. As an

example use the function peaks which generates a surface with a couple of peaks and valleys:

```
z = peaks;
surf(z), colormap(jet), colorbar
```

The colorbar indicates that the minimum element of z (somewhat less than −6) is mapped to row 1 of the colormap (R=0, G=0, B=0.5625), whereas the maximum element (about 8) is mapped to row 64 (R=0.5625, G=0, B=0).

You can specify the color with a second argument the same size as the first:

```
z = peaks(16);   % generates a 16-by-16 mesh
c = rand(16);
surf(z, c), colormap(prism)
```

Here the surface is tiled with a random pattern of 16-by-16 colors from the prism colormap.

In this form of surf each element of c is used to determine the color of the point in the corresponding element of z. By default MATLAB uses a process called *scaled mapping* to map from an element of z (or c) to the color in the colormap. The details of the scaling are determined by the caxis command. For further details see help caxis or the section **Coloring Mesh and Surface Plots** in **MATLAB Help: 3-D Visualization: Creating 3-D Graphs**.

You can exploit the facility to specify color in order to emphasize properties of the surface. The following example is given in the MATLAB documentation:

```
z = peaks(40);
c = del2(z);
surf(z, c)
colormap hot
```

The function del2 computes the discrete Laplacian of a surface—the Laplacian is related to the curvature of the surface. Creating a color array from the Laplacian means that regions with similar curvature will be drawn in the same color. Compare the surface obtained this way with that produced by the statement:

```
surf(P), colormap(hot)
```

In the second case regions with similar heights about the x-y plane have the same color.

Alternative forms of surf (and related surface functions) are:

```
surf(x, y, z)        % colour determined by z
surf(x, y, z, c)     % colour determined by c
```

9.6.3 Truecolor

The technique of coloring by means of a colormap is called *indexed* coloring—a surface is colored by assigning each data point an index (row) in the color map. The Truecolor technique colors a surface using explicitly specified RGB triplets. Here is another example from the MATLAB documentation (it also demonstrates the use of a multi-dimensional array):

```
z = peaks(25);
c(:,:,1) = rand(25);
c(:,:,2) = rand(25);
c(:,:,3) = rand(25);
surf(z, c)
```

The three "pages" of `c` (indicated by its third subscript) specify the values of RGB respectively to be used to color the point in `z` whose subscripts are the same as the first two subscripts in `c`. For example, the RGB values for the color of the point `z(5,13)` are given by `c(5,13,1)`, `c(5,13,2)`, and `c(5,13,3)` respectively.

9.7 LIGHTING AND CAMERA

MATLAB uses lighting to add realism to graphics, e.g., illuminating a surface by shining light on it from a certain angle. Here are two examples from the MATLAB documentation. Have a look at them.

```
z = peaks(25);
c(:,:,1) = rand(25);
c(:,:,2) = rand(25);
c(:,:,3) = rand(25);
surf(z,c,'facecolor','interp','facelighting','phong',...
         'edgecolor','none')
camlight right
```

The possibilities for the `facelighting` property of the surface object are `none`, `flat` (uniform color across each facet), `gouraud` or `phong`. The last two are the names of lighting algorithms. Phong lighting generally produces better results, but takes longer to render than Gouraud lighting. Remember that you can see all a surface object's properties by creating the object with a handle and using `get` on the object's handle.

This one's quite stunning:

```
[x y ] = meshgrid(-8 : 0.5 : 8);
r = sqrt(x.^2 + y.^2) + eps;
z = sin(r) ./ r;
```

```
surf(x,y,z,'facecolor','interp','edgecolor','none', ...
             'facelighting','phong')
colormap jet
daspect([5 5 1])
axis tight
view(-50, 30)
camlight left
```

For more information on lighting and the camera see the sections **Lighting as a Visualization Tool** and **Defining the View** in **MATLAB Help: 3-D Visualization**.

9.8 SAVING, PRINTING AND EXPORTING GRAPHS

9.8.1 Saving and opening figure files

You can save a figure generated during a MATLAB session so that you can open it in a subsequent session. Such a file has the `.fig` extension.

- Select **Save** from the figure window **File** menu.
- Make sure the **Save as type** is `.fig`.

To open a figure file select **Open** from the **File** menu.

9.8.2 Printing a graph

You can print everything inside a figure window frame, including axis labels and annotations:

- To print a figure select **Print** from the figure window **File** menu.
- If you have a black and white printer, colored lines and text are "dithered to gray" which may not print very clearly in some cases. In that case select **Page Setup** from the figure **File** menu. Select the **Lines and Text** tab and click on the **Black and white** option for **Convert solid colored lines to:**.

9.8.3 Exporting a graph

A figure may be exported in one of a number of graphics formats if you want to import it into another application, such as a text processor. You can also export it to the Windows clipboard and paste it from there into an application.

To export to the clipboard:

- Select **Copy Figure** from the figure window's **Edit** menu (this action copies to the clipboard).

- Before you copy to the clipboard you may need to adjust the figure's settings. You can do this by selecting **Preferences** from the figure's **File** menu. This action opens the **Preferences** panel, from which you can select **Figure Copy Template Preferences** and **Copy Options Preferences** to adjust the figure's settings.

  You may also have to adjust further settings with **Page Setup** from the figure's **File** menu.

  To export a figure to a file in a specific graphics format:

- Select **Export** from the figure's **File** menu. This action invokes the **Export** dialog box.
- Select a graphics format from the **Save as type** list, e.g., EMF (enhanced metafiles), JPEG, and so on. You may need to experiment to find the best format for the target application you have in mind.

  For example, to insert a figure into a Word document, I find it much easier first to save it in EMF or JPEG format and then to insert the graphics file into the Word document, rather than to go the clipboard route (there are more settings to adjust that way).

For further details consult the section **Basic Printing and Exporting** in **MATLAB Help: Graphics**.

SUMMARY

- 2-D plots are drawn with the `plot` statement.
- There is a set of easy-to-use plotters called `ez*`.
- Graphs may be labeled with `grid`, `text`, `title`, `xlabel`, `ylabel`, and others.
- Multiple plots may be drawn on the same axes in a number of ways.
- Line styles, markers and color may be varied.
- Axis limits may be set explicitly.
- In MATLAB the word "axes" refers to the graphics object in which the x and y axis and their labels, plots and text annotations are drawn.
- A number of axes may be drawn in the same figure with `subplot`.
- Co-ordinates of points in a figure may be selected with `ginput`.
- `semilogx`, `semilogy` and `loglog` are used to plot on $\log_{10}$ scales.
- The `polar` command plots in polar co-ordinates.
- `fplot` provides a handy way of plotting mathematical functions.
- `plot3` draws lines in 3-D.
- `comet3` animates a 3-D plot.
- A 3-D surface may be plotted with `mesh`.
- A 3-D plot may be rotated with `view` or with the **Rotate 3-D** tool in the figure window.

- `mesh` may also be used to visualize a matrix.
- `contour` and `countour3` draws contour levels in 2-D and 3-D respectively.
- 3-D surfaces may be cropped.
- For a complete list of graphics functions consult the online Help in **MATLAB Function Reference: Functions by Category: Graphics**.
- MATLAB graphics objects are arranged in a parent-child hierarchy.
- A handle may be attached to a graphics object at creation; the handle may be used to manipulate the graphics object.
- If `h` is the handle of a graphics object, `get(h)` returns all the current values of the object's properties. `set(h)` shows you all the possible values of the properties.
- The functions `gcf`, `gca` and `gco` return the handles of various graphics objects.
- Use `set` to change the properties of a graphics object.
- A graph may be edited to a limited extent in plot edit mode, selected from the figure window (**Tools -> Edit Plot**). More general editing can be done with the Property Editor (**Edit -> Figure Properties** from the figure window).
- The most versatile way to create animations is by using the Handle Graphics facilities. Other techniques involve comet plots and movies.
- Indexed coloring may be done with colormaps.
- Graphs saved to `.fig` files may be opened in subsequent MATLAB sessions.
- Graphs may be exported to the Windows clipboard, or to files in a variety of graphics formats.

CHAPTER EXERCISES

9.1 Draw a graph of the population of the USA from 1790 to 2000, using the (logistic) model:

$$P(t) = \frac{197,273,000}{1 + e^{-0.03134(t-1913.25)}}$$

where t is the date in years.

Actual data (in 1000s) for every decade from 1790 to 1950 are as follows: 3929, 5308, 7240, 9638, 12,866, 17,069, 23,192, 31,443, 38,558, 50,156, 62,948, 75,995, 91,972, 105,711, 122,775, 131,669, 150,697. Superimpose this data on the graph of $P(t)$. Plot the data as discrete circles (i.e., do not join them with lines) as shown in Figure 9.16.

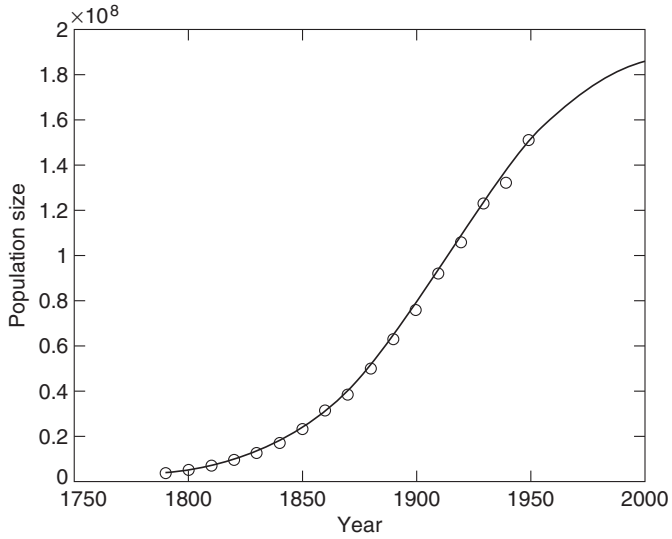

FIGURE 9.16 USA population: model and census data (o).

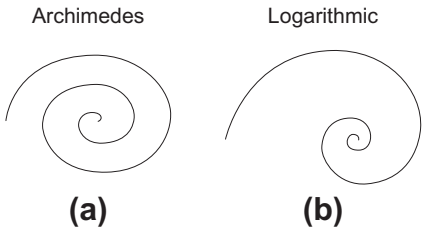

FIGURE 9.17 Spirals.

9.2 The Spiral of Archimedes (Figure 9.17) may be represented in polar co-ordinates by the equation:

$$r = a\theta,$$

where a is some constant. (The shells of a class of animals called nummulites grow in this way.) Write some command-line statements to draw the spiral for some values of a.

9.3 Another type of spiral is the *logarithmic* spiral (Figure 9.17), which describes the growth of shells of animals like the periwinkle and the nautilus. Its equation in polar co-ordinates is:

$$r = aq^{\theta},$$

where $a>0$, $q>1$. Draw this spiral.

9.4 The arrangement of seeds in a sunflower head (and other flowers, like daisies) follows a fixed mathematical pattern. The nth seed is at position:

$$r = \sqrt{n},$$

with angular co-ordinate $\pi dn/180$ radians, where d is the constant angle of divergence (in degrees) between any two successive seeds, i.e., between the nth and $(n+1)$th seeds. A perfect sunflower head (Figure 9.18) is generated by $d=137.51°$. Write a program to plot the seeds; use a circle (o) for each seed. A remarkable feature of this model is that the angle d must be exact to get proper sunflowers. Experiment with some different values, e.g., 137.45° (spokes, from fairly far out), 137.65° (spokes all the way), 137.92° (Catherine wheels).

9.5 The equation of an ellipse in polar co-ordinates is given by:

$$r = a(1 - e^2)/(1 - e\cos\theta),$$

where a is the semi-major axis and e is the eccentricity, if one focus is at the origin, and the semi-major axis lies on the x-axis.

Halley's Comet, which visited us in 1985/6, moves in an elliptical orbit about the Sun (at one focus) with a semi-major axis of 17.9 A.U. (A.U. stands for Astronomical Unit, which is the mean distance of the Earth from the Sun: 149.6 million km.) The eccentricity of the orbit is 0.967276. Write a program which draws the orbit of Halley's Comet and the Earth (assume the Earth is circular).

9.6 A very interesting iterative relationship that has been studied a lot recently is defined by:

$$y_{k+1} = ry_k(1 - y_k)$$

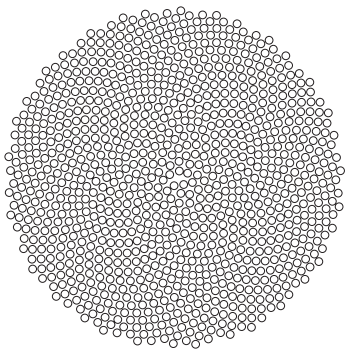

FIGURE 9.18 A perfect sunflower?

(this is a discrete form of the well-known *logistic model*). Given y_0 and r, successive y_ks may be computed very easily, e.g., if $y_0 = 0.2$ and $r = 1$, then $y_1 = 0.16, y_2 = 0.1334$, and so on.
This formula is often used to model population growth in cases where the growth is not unlimited, but is restricted by shortage of food, living area, amongst other things.
y_k exhibits fascinating behavior, known as *mathematical chaos*, for values of r between 3 and 4 (independent of y_0). Write a program which plots y_k against k (as individual points).
Values of r that give particularly interesting graphs are 3.3, 3.5, 3.5668, 3.575, 3.5766, 3.738, 3.8287, and many more that can be found by patient exploration.

9.7 A rather beautiful *fractal* picture can be drawn by plotting the points (x_k, y_k) generated by the following difference equations:

$$x_{k+1} = y_k(1 + \sin 0.7x_k) - 1.2\sqrt{|x_k|},$$
$$y_{k+1} = 0.21 - x_k,$$

starting with $x_0 = y_0 = 0$. Write a program to draw the picture (plot individual points; do not join them).