Introduction to MATLAB Programming

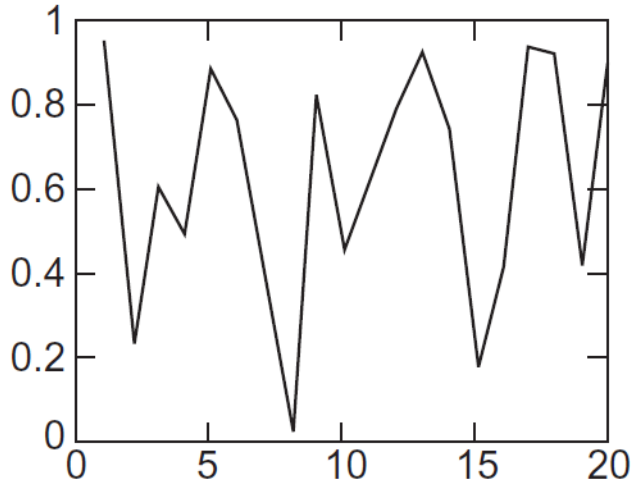
Graph

Basic 2-D Graphics (1)

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.

Graphs (in 2-D) are drawn with the `plot` statement. In its simplest form, it takes a single vector argument as in `plot(y)`.

```
plot(rand(1, 20))
```



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

Basic 2-D Graphics (2)

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

```
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.

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])
```

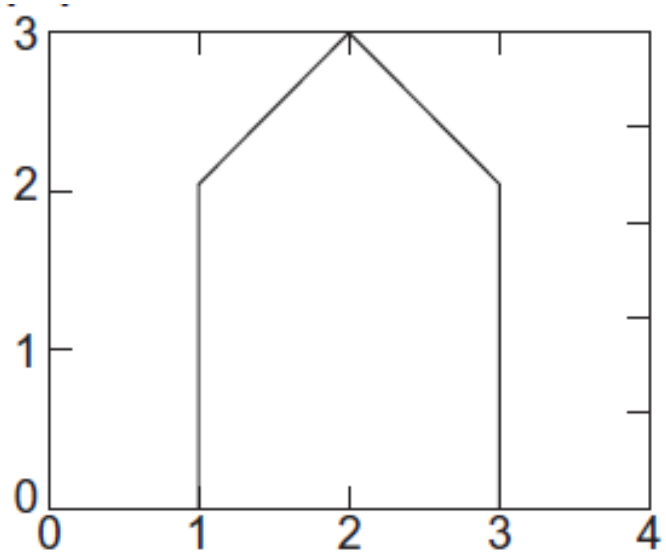
i.e., `[0 4]` contains the x co-ordinates of the two points, and `[1 3]` contains their y co-ordinates.

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"



Labels

Graphs may be labeled with the following statements:

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

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.

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' )
```

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.

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.

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.

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.

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`.

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
```

`axes and axis?`

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.

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.

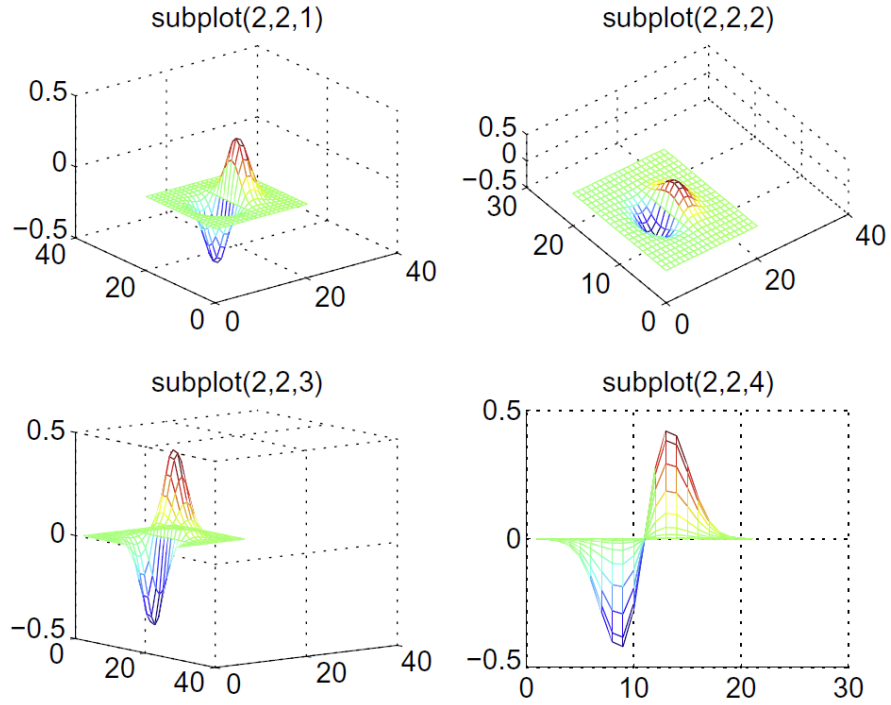


FIGURE 9.2 Four subplots: rotations of a 3-D surface.

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.

`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.

Logarithmic plots

The command:

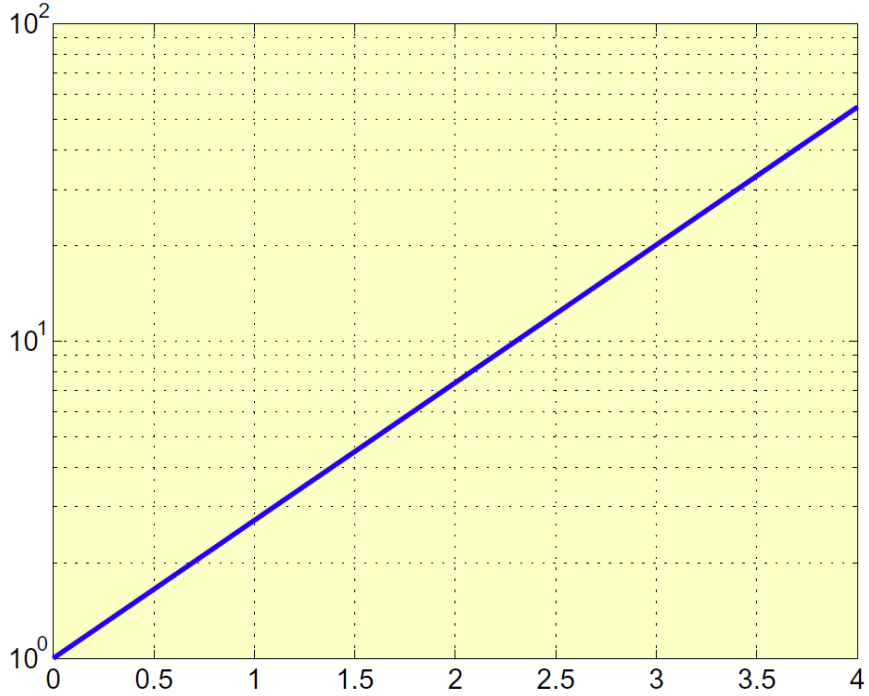
```
semilogy(x, y)
```

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`.



9.3 A logarithmic plot.

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

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:

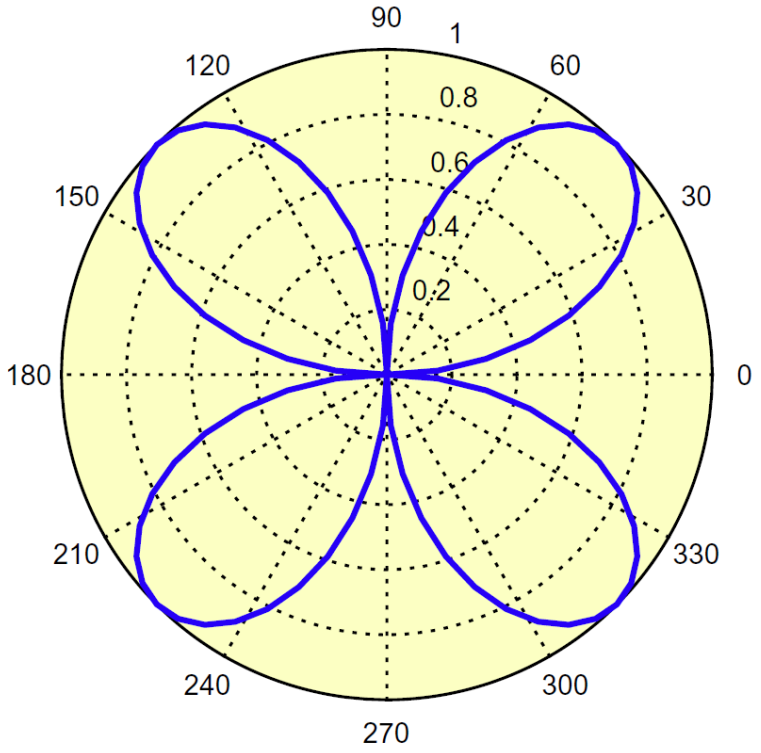
```
polar(theta, r)
```

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.



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.

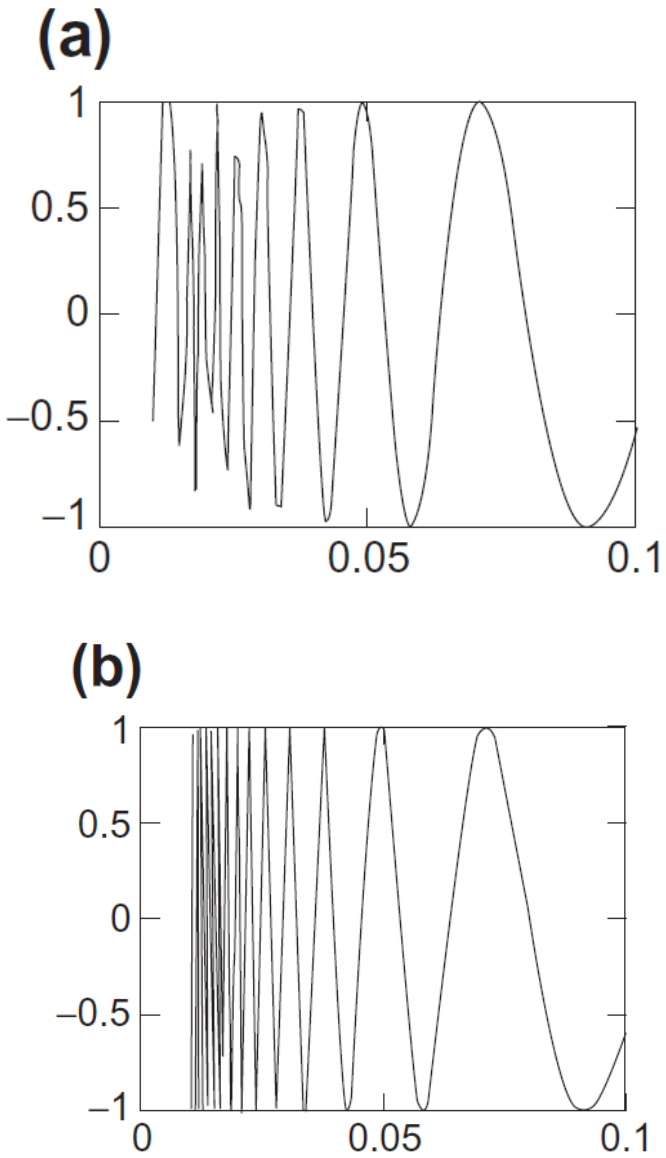
For example, the statements:

```
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!
```



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.

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.

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`.

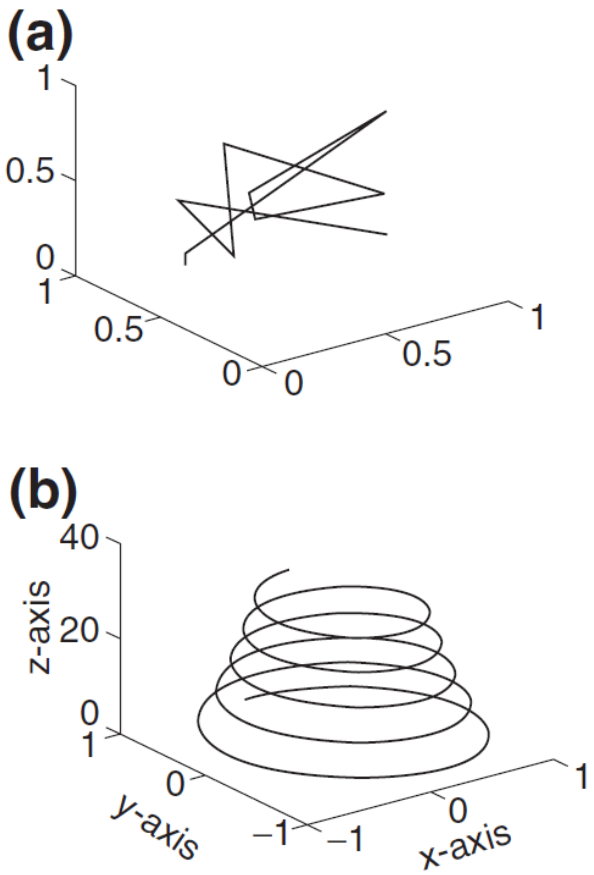


FIGURE 9.6 Examples of `plot3`.

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.

```
t=-10*pi:pi/250:10*pi;
 x=(cos(2*t).^2).*sin(t);
y=(sin(2*t).^2).*cos(t);
comet3(x,y,t)
```

Mesh surfaces (1)

```
[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.)

Mesh surfaces (2)

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
```

Mesh surfaces (3)

```
[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.)

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:

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

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.

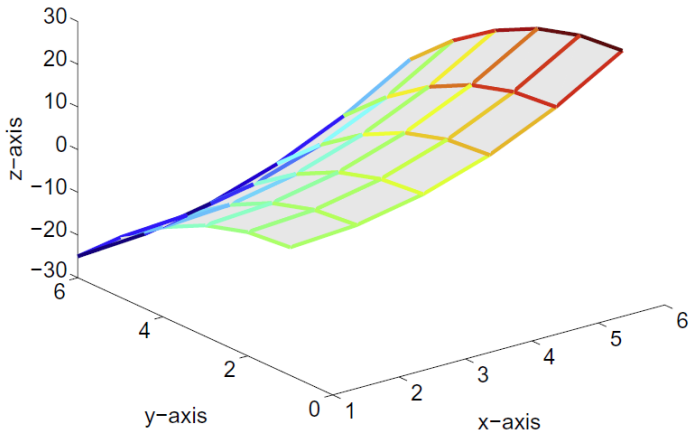


FIGURE 9.7 The surface $z = x^2 - y^2$.

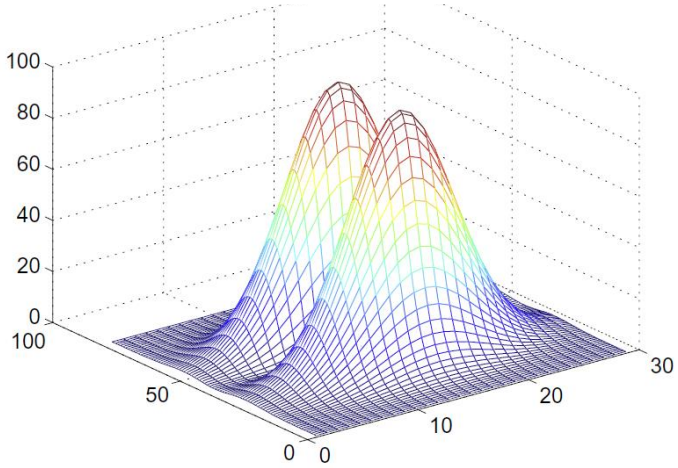


FIGURE 9.8 Heat distribution over a steel plate.

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)
```

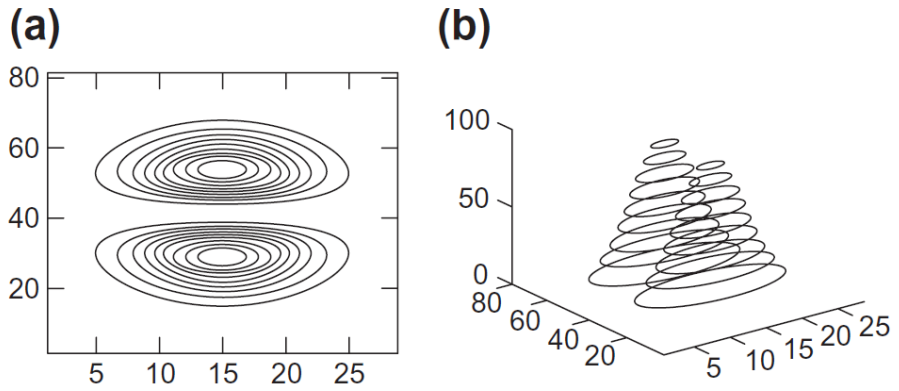


FIGURE 9.9 Contour plots.

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.

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.

Cropping a surface with NaNs

If a matrix for a surface plot contains NaNs, 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.

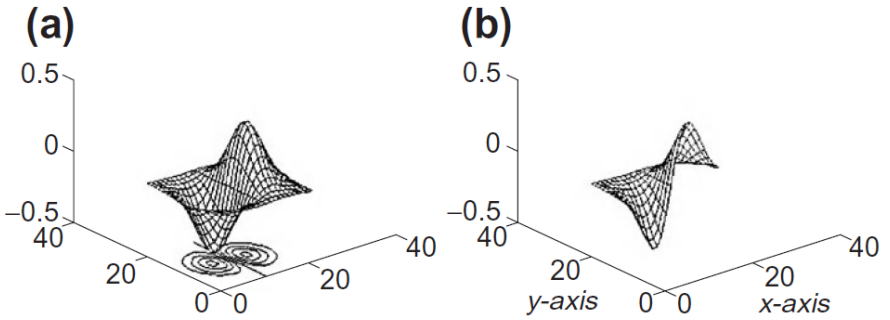


FIGURE 9.10 (a) `meshc`; (b) cropping a surface.

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`.

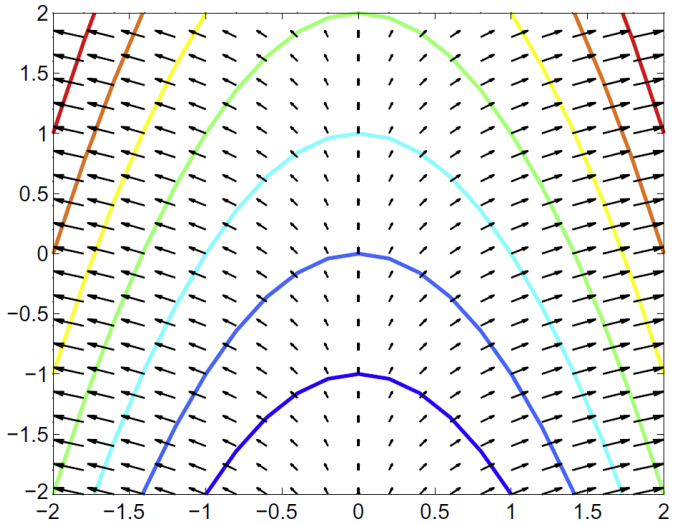


FIGURE 9.11 Gradients and level surfaces.

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.

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.

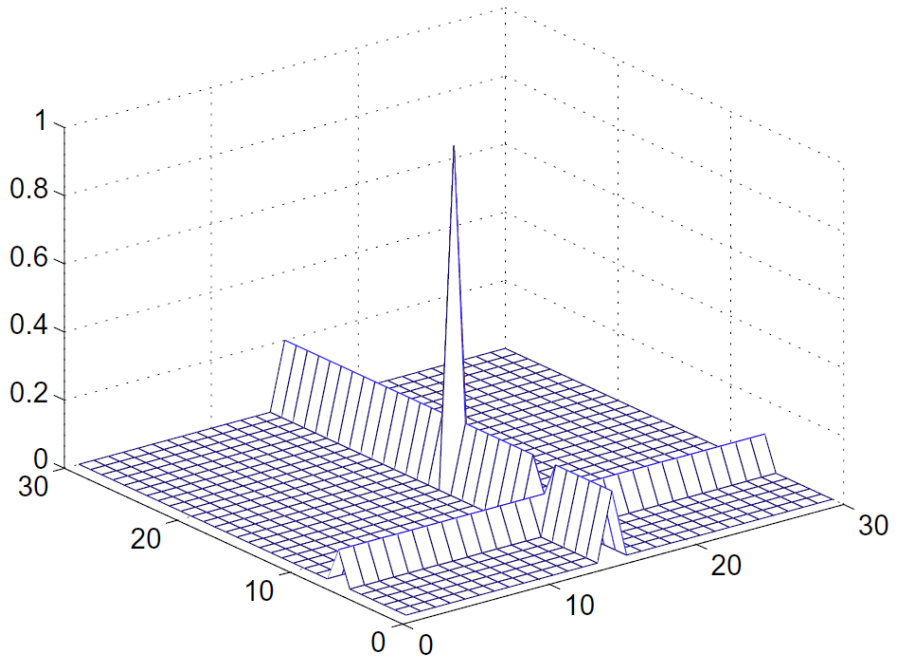


FIGURE 9.12 Visualization of a matrix.

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°.

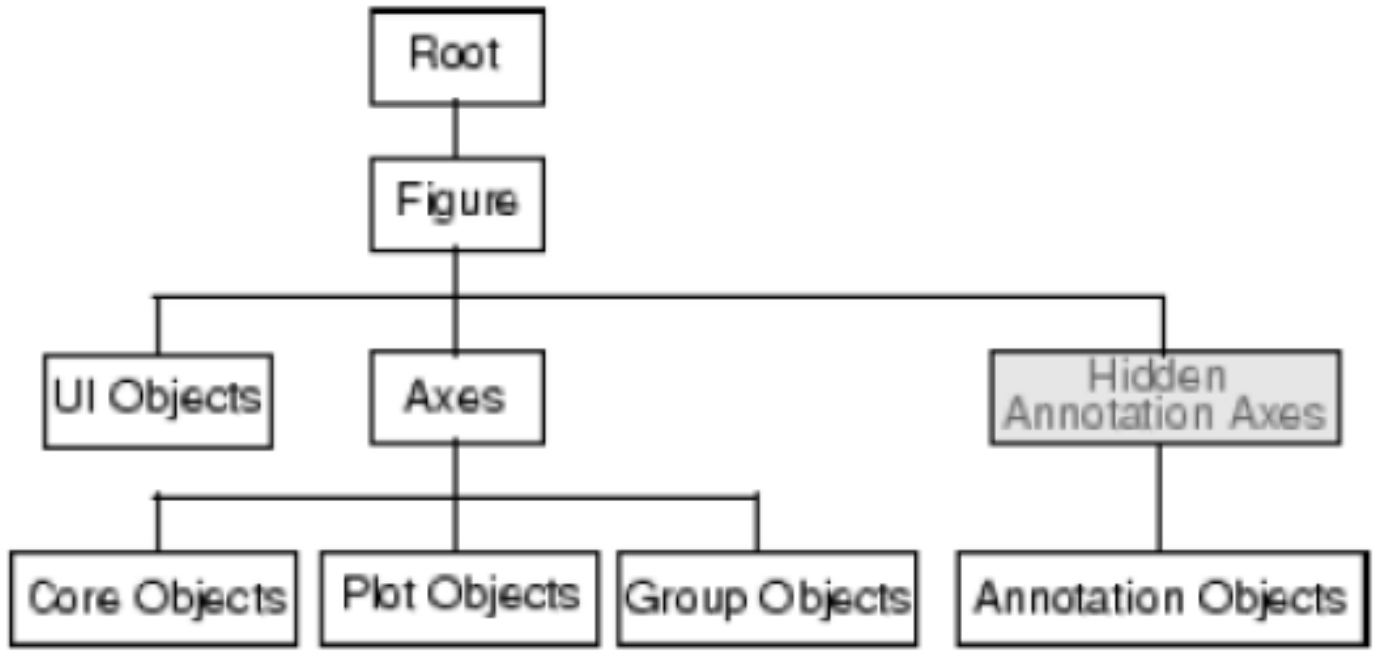
The command `pause(n)` suspends execution for `n` seconds.

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.

Handle graphics (1)

- "*Graphics Objects*" are the basic drawing elements used by MATLAB to display data.
- "*Graphics Handles*" are kind of "ID" to identify each graphic object such as figure, individual plot lines, surface, legend, text, etc.
- Using the handles, we can manipulate characteristics of each graphics object. This give us tremendous control of figure parameters.
- The graphic objects are arranged in "parent-children" relationship.

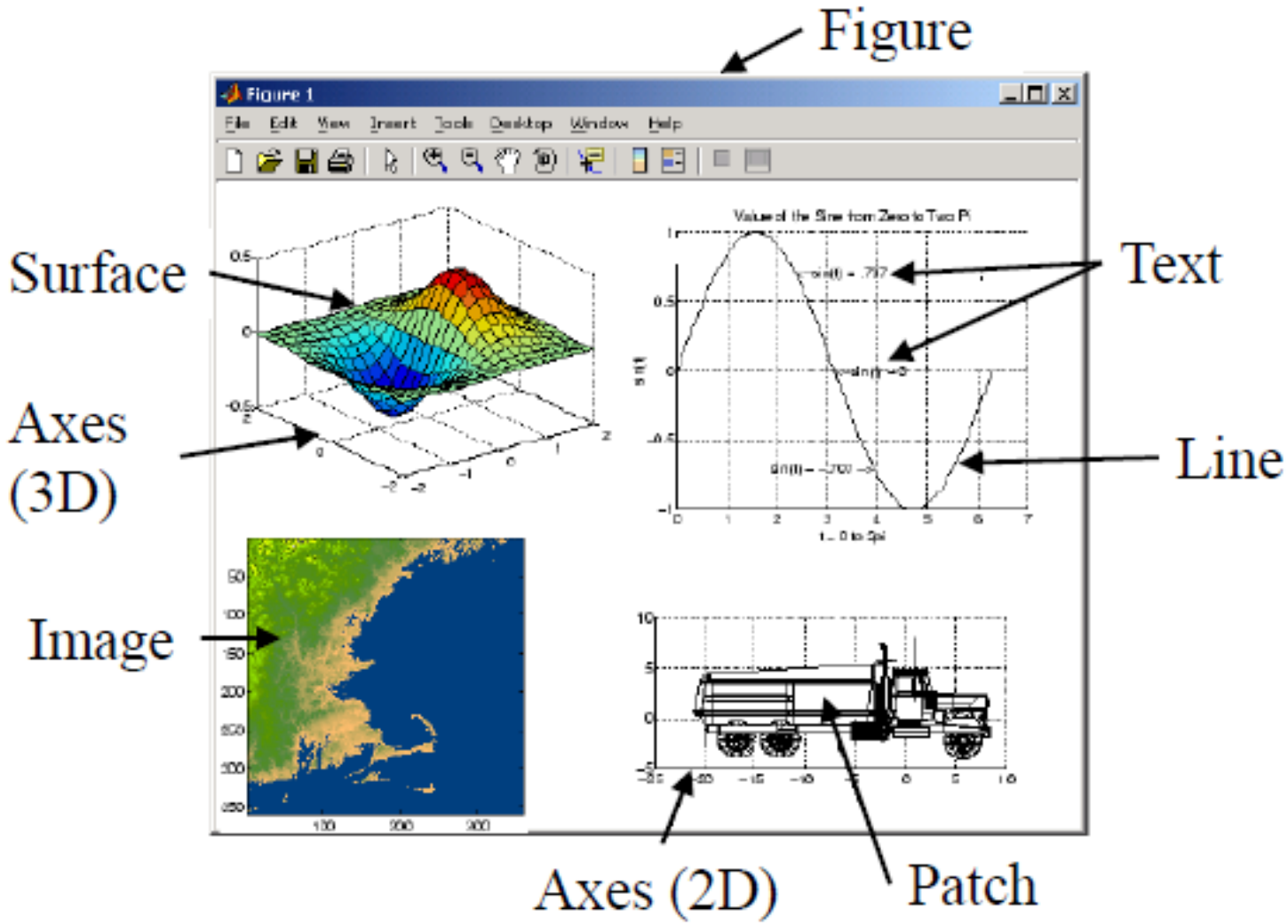
Hierarchy of Graphics Objects



Note: 1. Axes object is a child of Figure object and so on..

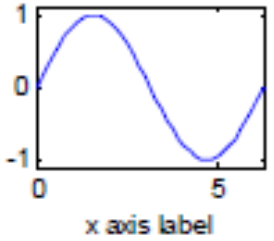
2. Line, Surface, Text objects are children of an Axes object

Graphics Objects



Graphics Handles

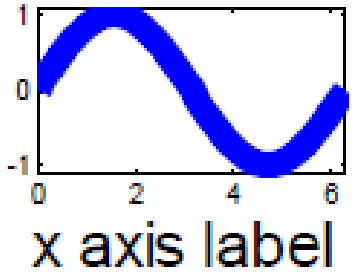
- All graphic objects have a handle that can be used to modify them
- For example:




```
>> x = 0:pi/20:2*pi;
>> hsin = plot(x,sin(x)) %Plots and returns a handle
>> hx = xlabel('x axis label') %Returns xlabel handle
```

- Using its handle, object properties can be modified

```
>> set(hsin, 'LineWidth', 10); % Increase line width
>> set(hx,'FontSize', 24) % Change Font Size of xlabel
```



Finding Available Graphics Properties

- To get a complete list of an object's properties, use the **`get( )`** function along with the object's handle.
- For example **`get(hsin)`** returns over 30 properties for the line object that has the handle **`hsin`** as shown.
- Notice that one of them is **`Color: [0 0 1]`**
- **`set( )`** can then be used to modify the line's color

```
>> set (hsin, 'color', [1   0   0  ]) % red
>> set (hsin, 'color', [0   1   0  ]) % green
>> set (hsin, 'color', [0   0   1  ]) % blue
>> set (hsin, 'color', [0.5 0.5 0.5]) % grey
```

etc.

- Note: the three elements in the color array define the ratio of red, green, and blue (a.k.a. RGB)

```
    DisplayName: ''
     Annotation: [1x1 hg.Annotation]
          Color: [0 0 1]
      LineStyle: '-'
      LineWidth: 10
         Marker: 'none'
     MarkerSize: 6
 MarkerEdgeColor: 'auto'
 MarkerFaceColor: 'none'
          XData: [1x41 double]
          YData: [1x41 double]
          ZData: [1x0 double]
    BeingDeleted: 'off'
  ButtonDownFcn: []
       Children: [0x1 double]
       Clipping: 'on'
       CreateFcn: []
       DeleteFcn: []
     BusyAction: 'queue'
 HandleVisibility: 'on'
        HitTest: 'on'
  Interruptible: 'on'
       Selected: 'off'
 SelectionHighlight: 'on'
            Tag: ''
           Type: 'line'
  UIContextMenu: []
       UserData: []
        Visible: 'on'
         Parent: 173.0503
    XDataMode: 'manual'
  XDataSource: ''
  YDataSource: ''
  ZDataSource: ''
```

Useful Functions to Get Handles

- If a handle is not known, here are a few functions that can be used to get it
  - **`gcf`** gets the handle of the current figure
  - **`gca`** gets the handle of the current axes

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:

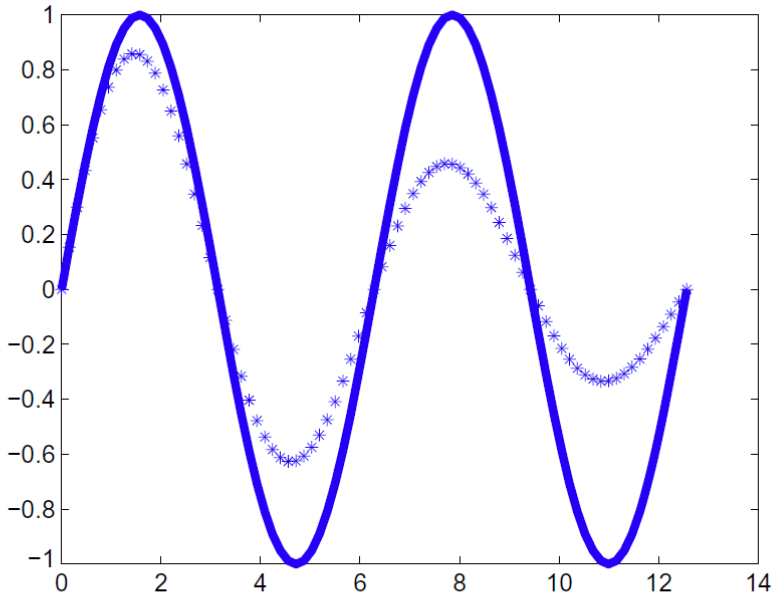


FIGURE 9.14 Result of manipulating a figure using the handles of axes children.

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

Editing plots

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.

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!

Animation

- To animate a plot, simply generate a series of snapshots and then use "move" to show them
- Example, animate sin(x)*sin(2*pi*t/20)
- Get file ***anim.m***

Animation Example

```
x=0:pi/100:2*pi;
y=sin(x);
plot(x,y)
axis tight
% Record the movie
for j = 1:20
    plot(x,sin(2*pi*j/20)*y)
    F(j) = getframe;
end
% Play the movie two times
movie(F,2)
```

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.

Animation with handle graphics

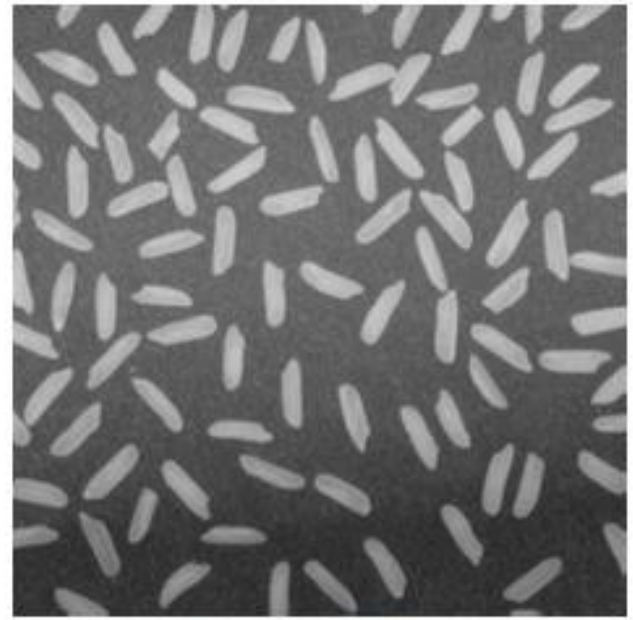
For starters, run the following script, which should show the marker o tracing out a sine curve, leaving a trail behind it:

```
% 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
```

MATLAB Image Processing Applications

Image Creation/Manipulation

Collages, etc.



Machine Vision

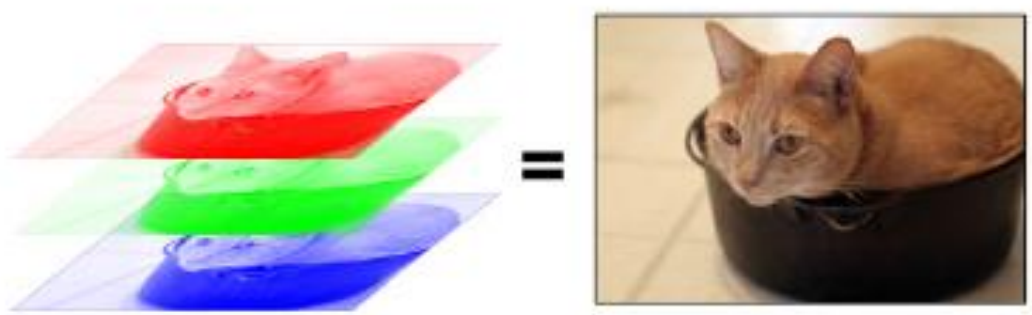
1. Counting
2. Measurements (size, area, color)
3. Defect Identification

Image Basics

- In the MATLAB workspace, most images are represented as two-dimensional arrays (matrices), in which each element of the matrix corresponds to a single pixel in the displayed image.
- Thus, MATLAB can manipulate images at the pixel level

RGB Images

- RGB images are 3D arrays composed of a stack of three 2D arrays (layers)
- The arrays (layers) are a maps of red, green, and blue intensities that, when combined, describe the final image



Supported Image Formats

- JPEG (Joint Photographic Experts Group)
- GIF (Graphics Interchange Files)
- PNG (Portable Network Graphics)
- TIFF (Tagged Image File Format)
- BMP (Microsoft® Windows® Bitmap)
- PCX (Paintbrush)
- Others

JPG vs. PNG

JPG：340KB

PNG：1.84MB

BMP vs. JPEG

BMP：500KB

JPG：184KB

Basic Image Functions

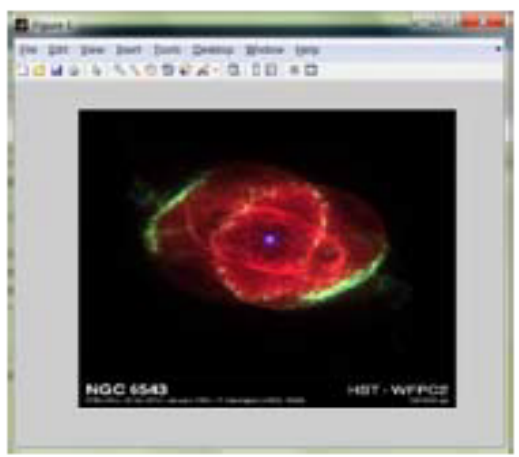
Function	Description
`image( )`	Display image object
`imshow( )`	Display image object or image file
`imread( )`	Read image from graphics file
`imwrite( )`	Write image to graphics file
`imfinfo( )`	Information about graphics file
`imagesc( )`	Scale data and display image object
`frame2im( )`	Return image data associated with movie frame

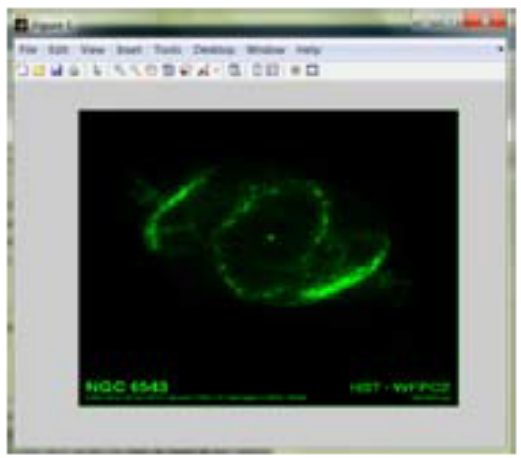
Image Example

```
A = imread('ngc6543a.jpg'); % Create image object
"A" from file
image(A); % Display image in figure
axis off; % Turn off axes
greens = A;
greens(:,:,[1 3]) = 0; % Set red (1) and blue (3)
intensities to zeros, leaving green (2) alone
image(greens) % Display green layer
imwrite(greens,'galaxy.jpg')
axis equal
```

All pixel columns

All pixel rows





MATLAB and Animation/Videos

- MATLAB has many tools for importing, viewing, and manipulating videos
- Possible applications
  - Create custom animations for presentations
  - Import images and stitch them together to create a movie or animated gif
  - Extract a single image from a video
  - Anything else you can dream up…

Animation and Movies Methods

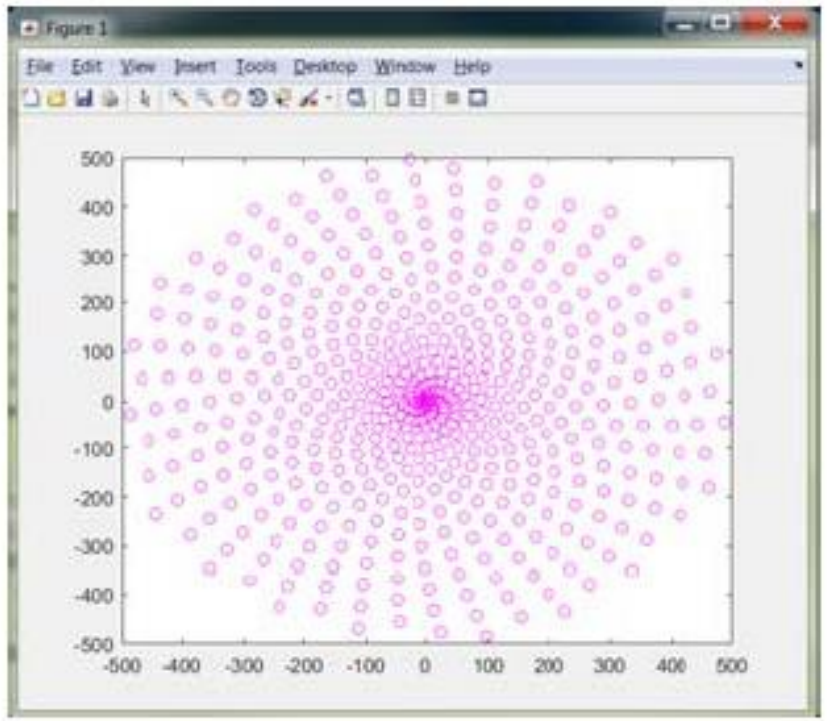
- Create a script using any of the 2D or 3D plot functions and the **`pause( )`** command
  - Can only "view/play" animation while inside MATLAB
- Create a movie object from a collection of images/plots
  - Can view/play in MATLAB AND export video file

Basic Animation Functions

Function	Description
`movie( )`	Play recorded movie frames
`getframe( )`	Capture movie frame from figure
`im2frame( )`	Convert image to movie frame
`VideoWriter( )`	Write videos to a file (avi, mpg, etc)
`VideoReader( )`	Import video file into MATLAB
`frame2im( )`	Return image data associated with movie frame
`pause(n)`	Pause for n seconds before continuing

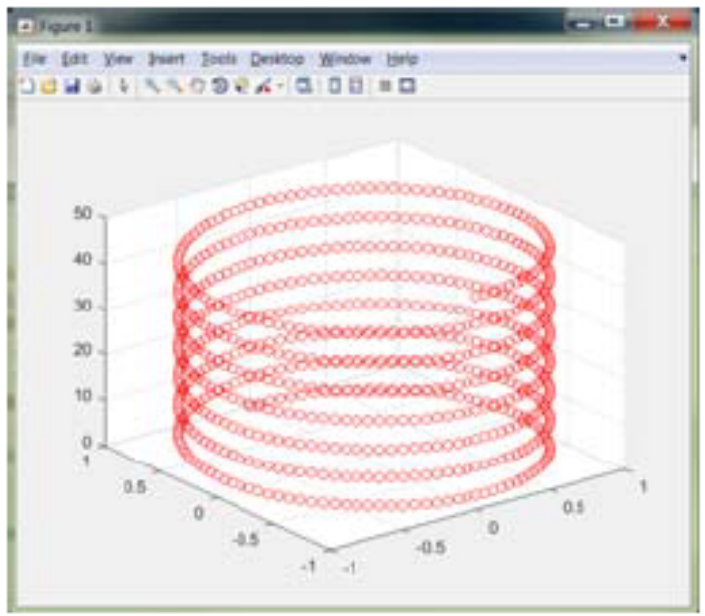
Animation Example with `pause( )`

```
clear all; close all; clc;
for t = 0:1:500
    x = t.*cos(t);
    y = t.*sin(t);
    plot(x,y,'mo')

    % keep previous plot point
    hold on

    % freeze axis size for
    % smooth animation
    axis([-500 500 -500 500]);

    % set time between frames
    % (limited by computer
    % capabilities)
    pause(0.005)
end
```



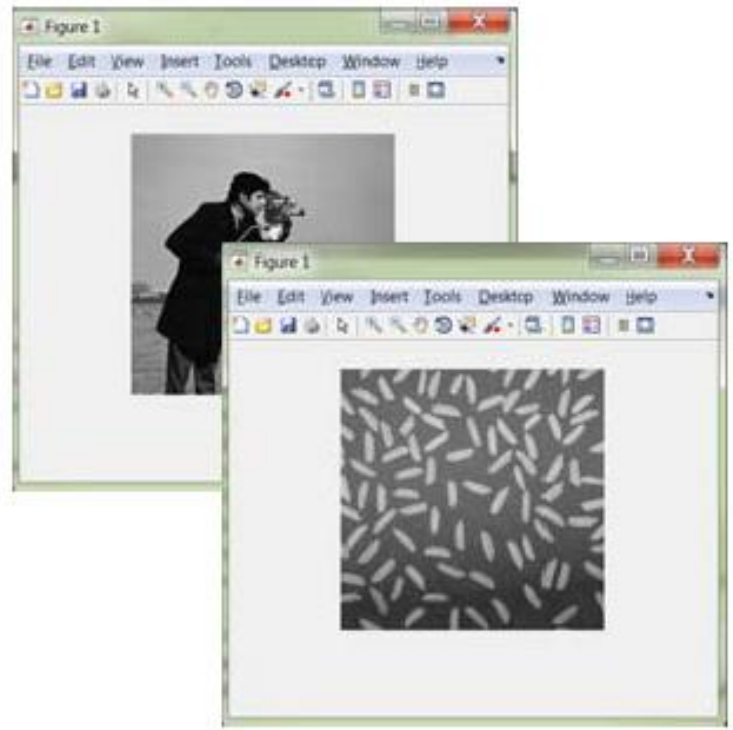
Animation Example with `pause( )`

```
%% Animation using plot3( ) and pause( )
clear all; close all; clc;
for t = 0:2*pi/100:15*pi
    x1 = sin(t);
    x2 = cos(t);
    plot3(x1,x2,t,'ro')
    hold on;  %turn this off and observe
    %keep axis size the same for each frame
    axis([-1 1 -1 1 0 50])
    axis off %optional
    grid on;
    %set time in seconds between frames
    %(limited by computer capabilities)
    pause(0.005) %modify to change animation speed
end
```



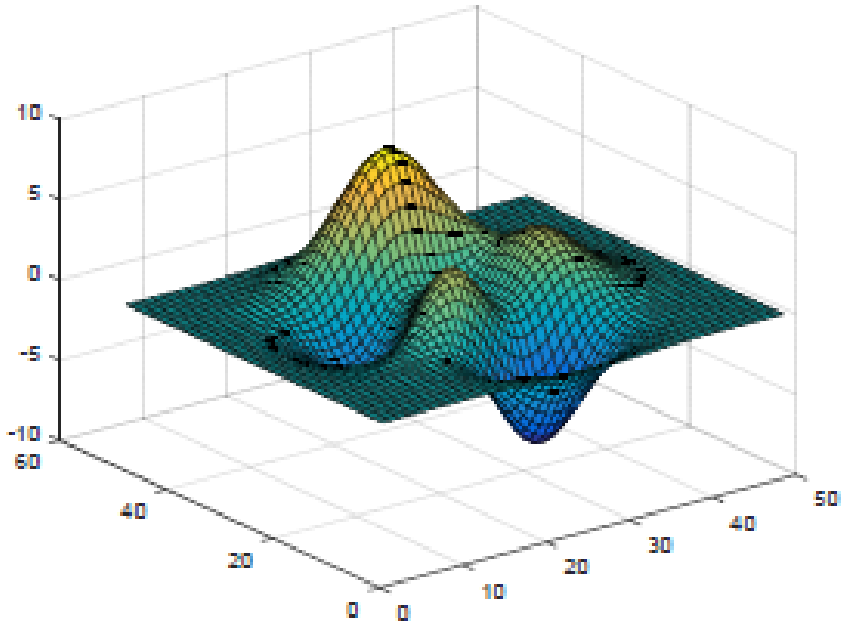
Animation Example with a Movie Object

```
clear all;close all;clc;
%display an image in current figure
imshow('cameraman.tif');
%convert the current figure to movie frame
M(1) = getframe; %add frame to movie object M
imshow('rice.png'); %disp a different image in current fig
M(2) = getframe; %add another frame to M
movie(M,5,2); %play movie M at 5 times at 2 frames per second (fps)
```



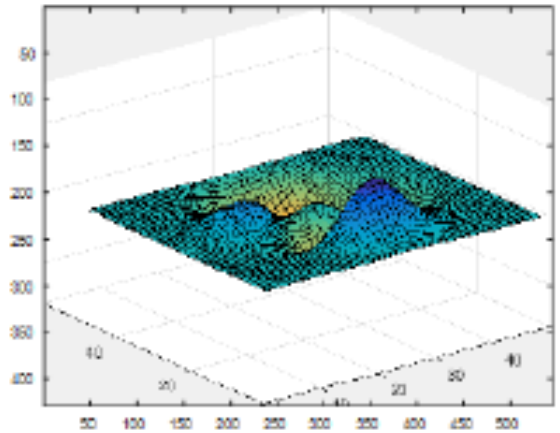
Animation Example with a Movie Object

```
clear all;close all;clc;
Z = peaks; %load data
surf(Z); %visualize data

% Initialize frame number
frameNum = 1;

for j = 1:20
  % Plot data in current figure
  surf(.01+sin(2*pi*j/20)*Z,Z);
  axis([0 50 0 60 -8 8])

  % Use current fig to create
  % a frame in movie object F
  F(frameNum) = getframe;

  % Increment frame number
  frameNum = frameNum + 1;
end
```



Extract Animation Frame and Play Movie Inside MATLAB

```
%% plot individual frame
close all;
%display frame 18 of movie object F
image(F(18).cdata)

%% play movie inside MATLAB
close all;
n = 5;    % number of times movie is repeated
fps = 6; % frames per second
         % (limited by cptr capabilities)
movie(F,n,fps)  %Play movie inside MATLAB
done = 1 % alert user when complete
```



Note: Movie object **F** needs to have been created beforehand

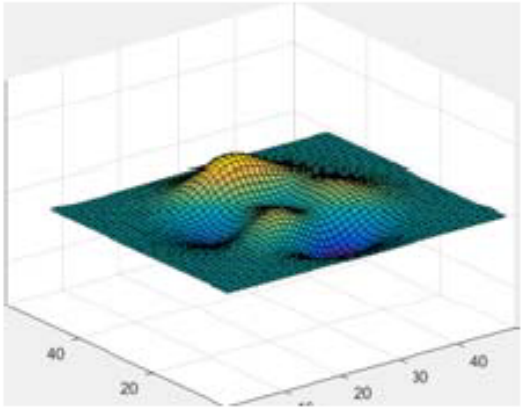
Writing Movie Object to .avi File

```
% Create VideoWriter object
writerObj = VideoWriter('Animation_Example.avi');

open(writerObj) % Open VideoWriter Object

% Loop through all frames of movie object F
for k = 1:length(F)
    % Write single frame from movie object F
    % to VideoWriter Object
    writeVideo(writerObj,F(k))
end
close(writerObj) %Close VideoWriter object
avidone = 1 %alert user when process is done
```

Writing Movie Object to an Animated .gif file

Because three-dimensional data is not supported for GIF files, call rgb2ind to convert the RGB data in the image to an indexed image A with a colormap map. To append multiple images to the first image, call imwrite with the name-value pair argument 'WriteMode','append'.
imwrite writes the GIF file to your current folder. Name-value pair 'LoopCount',Inf causes the animation to continuously loop.

```
filename = 'Animation_Example.gif';

% Loop through all frames of movie object F
for k = 1:length(F)
    % Extract image data of one frame of F
    im = frame2im(F(k));
    % Convert RGB image to indexed image
    [imind,cm] = rgb2ind(im,256);
    if k == 1; % Do this for the first frame
        % Write image to file
        imwrite(imind,cm,filename,'gif', 'Loopcount',inf);
    else % Do this for all the rest of the frames
        % Write image to file
        imwrite(imind,cm,filename,'gif','WriteMode','append');
    end
end

gifdone = 1 %alert user when gif has been written
```



Practice problem

Use **subplot** to show the difference between the **sin** and **cos** functions. Create an x vector with 100 linearly spaced points in the range from -2π to 2π, and then two y vectors for **sin(x)** and **cos(x)**. In a 2×1 **subplot**, use the **plot** function to display them, with appropriate titles.

Practice problem

Write a script that will plot the **sin** function three times in one Figure Window, using the colors red, green, and blue.

Practice problem

When an object with an initial temperature T is placed in a substance that has a temperature S, according to Newton's law of cooling, in t minutes it will reach a temperature T_t using the formula $T_t = S + (T - S)\, e^{(-kt)}$, where k is a constant value that depends on properties of the object. For an initial temperature of 100 and k = 0.6, graphically display the resulting temperatures from 1 to 10 minutes for two different surrounding temperatures: 50 and 20. Use the **plot** function to plot two different lines for these surrounding temperatures, and store the handle in a variable. Notice that two function handles are actually returned, and stored in a vector. Use **set** to change the line width of one of the lines.

Practice problem

Experiment with the **comet3** function: Try the example given when **help comet3** is entered and then animate your own function using **comet3**.

Practice problem

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.

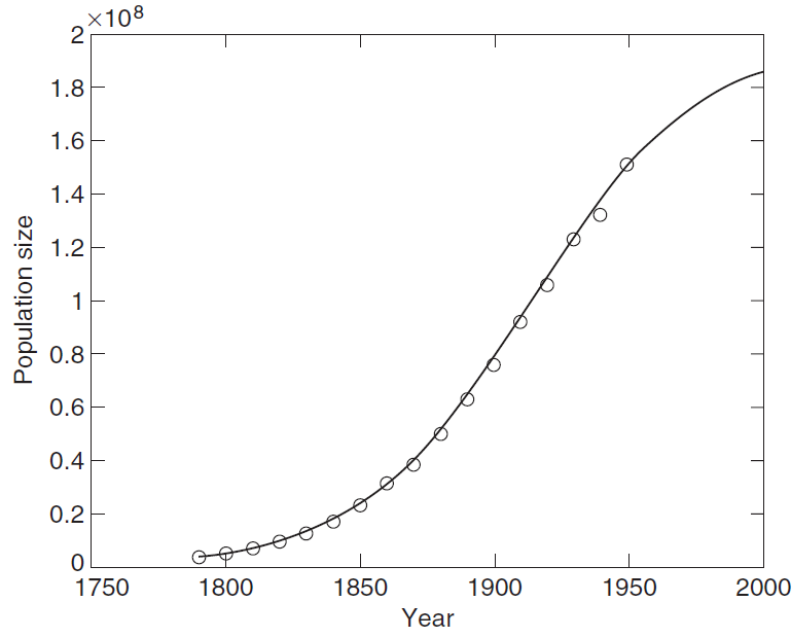


FIGURE 9.16 USA population: model and census data (○).

Practice problem

The Spiral of Archimedes (Figure 9.17) may be represented in polar coordinates 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.

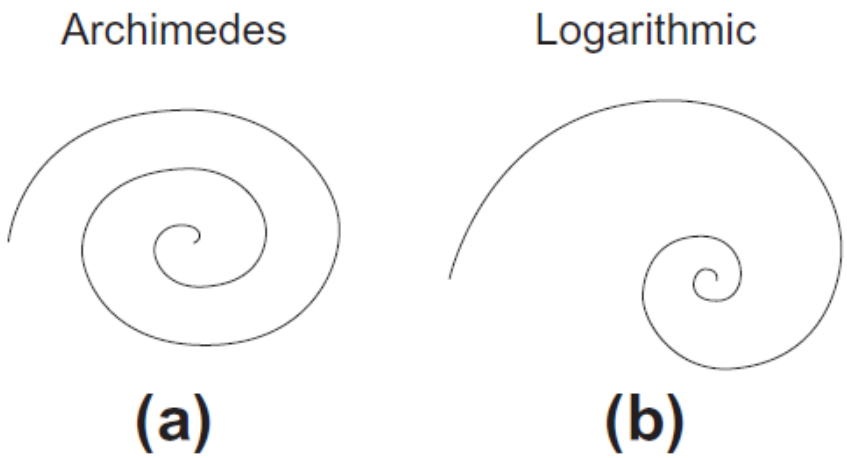


FIGURE 9.17 Spirals.

Practice problem

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.

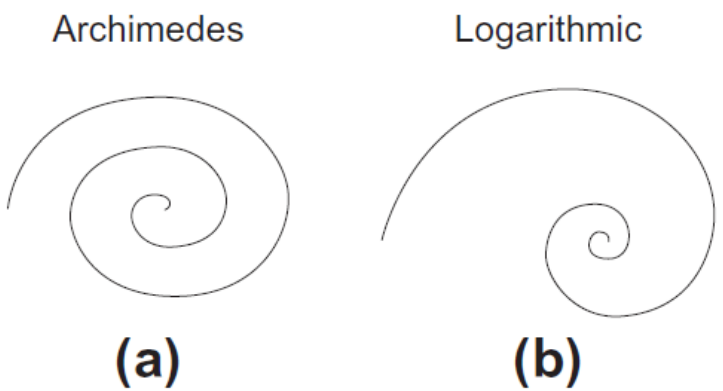


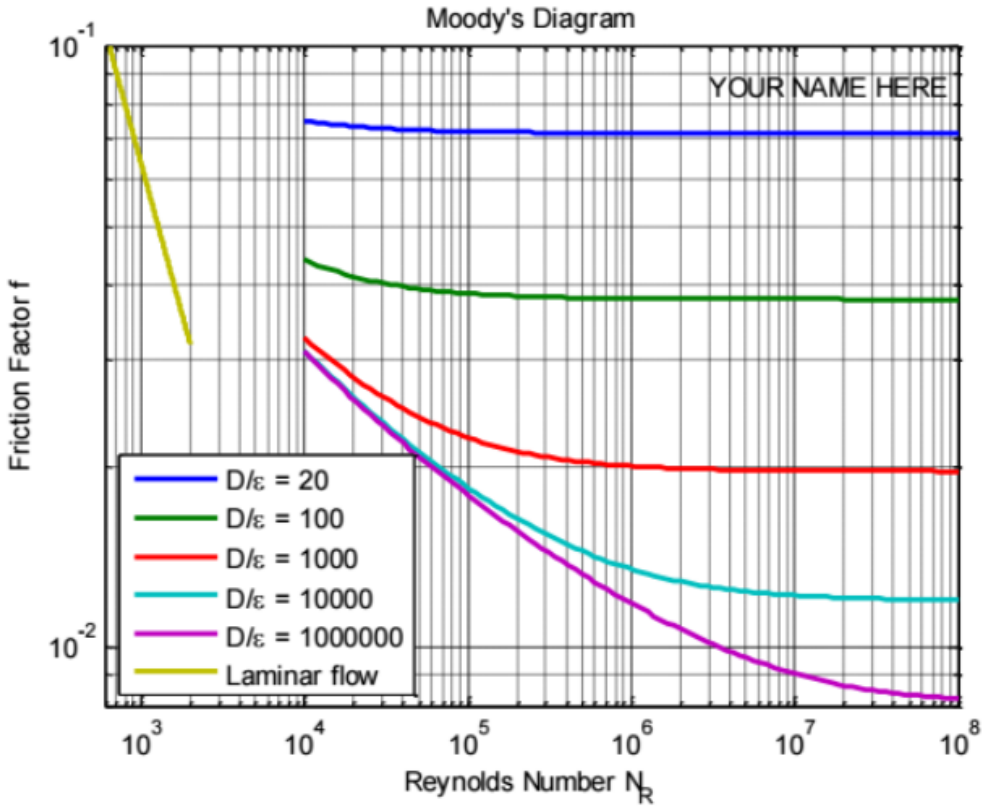
FIGURE 9.17 Spirals.

Practice problem

Moody's diagram as shown on the next page is a famous plot used to determine the effect internal friction (surface roughness) has on fluids flowing in pipes. The equation below has been developed by Jain and Swamee for the friction factor (f) for turbulent pipe flow.

$$f = \frac{0.25}{\left[\log\left(\frac{1}{3.7(D/\varepsilon)} + \frac{5.74}{N_R^{0.9}}\right)\right]^2}$$

a. Plot the friction factor following function for values Reynolds number (N_R) between 10^4 and 10^8 for D/ε values of 20, 100, 1000, 10,000, and 100,000. Your plot should look similar to example below. A couple of hints:
   a. Notice that both axes are log scaled
   b. log( ) in MATLAB does not do what you think it does
   c. The logspace( ) function may be a helpful in generating data that is evenly spaced on log-scaled axes (similar to linspace( )).
b. Add a line for $f = 64/N_R$ for smooth pipes to the same plot (be sure to match the N_R range shown in the figure).
c. Add a complete title and x and y axis labels (including any of the subscripts or superscripts).
d. Add a legend with the actual Greek character epsilon for each trace (e.g. D/ε = 1000, etc.). It can be in any position on the figure.
e. Adjust the axis limits so it looks like the figure on the last page.
f. Use the "text" command to print your name anywhere on the plot.



Practice problem

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).

Practice problem

The Wind Chill Factor (WCF) measures how cold it feels with a given air temperature T (in degrees Fahrenheit) and wind speed (V, in miles per hour). One formula for it is

$$WCF = 35.7 + 0.6T - 35.7\,(V^{0.16}) + 0.43T\,(V^{0.16})$$

Experiment with different plot types to display the WCF for varying wind speeds and temperatures.

Practice problem

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

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

(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.

Practice problem

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).

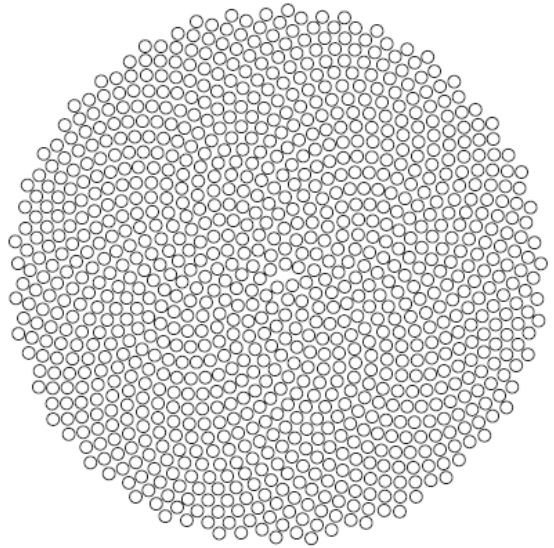


FIGURE 9.18 A perfect sunflower?

Practice problem

Place your name in the right corner as shown in the figure. For this assignment, the code for the plots are given and you are expected to create a user defined function that places your name in the corner of any plot.

```
%%
% This MATLAB script tests a user-defined function that places a name
% on randomly sized plots.

%% Generate randomized data to plot
clear all;clc;

% x data
xmin = (-10) + (10-(-10)).*rand; %Generate random number between -10
and 10
xrange = 2 + (5-2).*rand; %Generate random number between 2 and 5
xmax = xmin + xrange;
numPts = 150; %Number of data points
x = linspace(xmin,xmax,numPts);
x2 = x-0.2*xrange;

% y data
Amp = 0.5 + (2-0.5).*rand; %Generate random amplitude between 0.5 and 2
Freq = 0.5 + (1.5-0.5).*rand; %Generate random freq between 0.5 and 1.5
y = Amp*sin(2*pi*Freq*x);
y2 = 2*Amp*cos(2*pi*Freq*x2);

%% Plot data and test your function
r = 2; %number of subplot rows
c = 2; %number of subplot columns

subplot(r,c,1)
plot(x,y)
%****************************
%TYPE THE NAME OF YOUR FUNCTION HERE TO PUT YOUR NAME IN THE UPPER LEFT
```