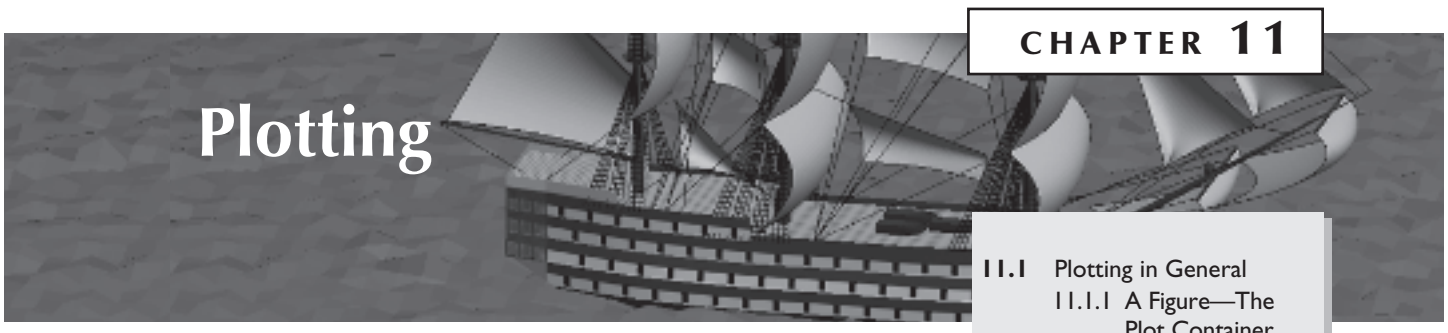
CHAPTER 11

Plotting



Chapter Objectives

This chapter presents the principles and practice of plotting in the following forms:

- Basic two-dimensional (2-D) line plots
- 2-D parametric plots
- Three-dimensional (3-D) line and parametric plots
- Basic 3-D surface plots
- Parametric surface plots
- Bodies of rotation

There is a much-quoted expression that "a picture is worth a thousand words," and this is never more appropriate than when talking about data. In previous chapters, we used some simple plot commands to display data to illustrate its behavior. The capability of the MATLAB language to present data reaches far beyond ordinary data plotting, and far beyond the limited confines of a textbook. This chapter will present the fundamental concepts of the different forms in which data can be presented, but it leaves to the reader the challenge of exploring the full range of capabilities available. You only really discover the power inherent in the plotting capabilities of MATLAB when you have some unusual data to visualize.

11.1 Plotting in General

Before considering the details of how each plotting mode works, we should set the context. In this section, we will discuss the general container for all graphical types, the figure, and some basic operations that apply to all figures—functions that enhance them, the ability to assemble subplots into a single figure, and the advisability of making manual changes to plots.

11.1.1 A Figure—The Plot Container

The fundamental container for plotting is a figure. In a simple script, if you just start plotting data, *figure number 1* is automatically generated to present the data. You can manage the figures by asserting the `figure` command. Each time `figure` is called, a new figure is made available, with the next higher figure number. If you use the form `figure <number>`, you can select a specific figure for the next plot.

To clear the current figure, put the key word `clf` in the header of your script. To remove all the figures, put the key phrase `close all` at the beginning of your script. The listing examples below will assume that each script begins with `clear, clc, close all`.

11.1.2 Simple Functions for Enhancing Plots

We have already introduced `plot(x, y)`, the basic function that creates a simple plot of `x` versus `y`. The following functions can be used to enhance any of the plots discussed in this chapter. Note that they enhance an existing plot; they should all be called after the fundamental function that creates a plot figure.

- `axis <param>` provides a rich set of tools for managing the appearance of the axes, including the following:
  - `tight` reduces the axes to their smallest possible size
  - `equal` sets the x and y scales to the same value
  - `square` makes the plot figure of equal width and height
  - `off` does not show the axes at all
- `axis([xl xu yl yu zl zu])` overrides the automatic computation of the axis values, forcing the x-axis to reach from `xl` to `xu`, the y-axis from `yl` to `yu`, and the z-axis from `zl` to `zu`. For 2-D plots, the z values should be omitted.
- `colormap <specification>` establishes a sequence of colors, the color map, to be used under a number of circumstances to cycle through a series of colors automatically. The legal specification values are listed in Appendix A.
- `grid on` puts a grid on the plot; `grid off` (the default) removes grid lines.

- `hold on` holds the existing data on the figure to allow subsequent plotting calls to be added to the current figure without first erasing the existing plot; `hold off` (the default) redraws the current figure, erasing the previous contents.
- `legend(...)` takes a cell array of strings, one for each of the multiple plots on a single figure, and creates a legend box. By default, that box appears in the top-right corner of the figure. However, this default can be overridden by explicitly specifying the location of the legend. See the help files for a complete discussion of the legend options.
- `shading <spec>` defines the method for shading surfaces. See the help files for a complete discussion of the shading specification options.
- `text(x, y, {z,}, str)` places the text provided at the specified (x, y) location on a 2-D plot, or at the (x, y, z) location on a 3-D plot.
- `title(...)` places the text provided as the title of the current plot.
- `view(az, el)` sets the angle from which to view a plot. The parameters are `az`, the azimuth, an angle measured in the horizontal plane, and `el`, the elevation, an angle measured upward from the horizontal. Both angles are specified in degrees.
- `xlabel(...)` sets the string provided as the label for the x-axis.
- `ylabel(...)` sets the string provided as the label for the y-axis.
- `zlabel(...)` sets the string provided as the label for the z-axis. (As we will see, all plots actually have a third axis.)

11.1.3 Multiple Plots on One Figure—Subplots

Within the current figure, you can place multiple plots with the `subplot` command, as shown in Figure 11.1. The function `subplot(r, c, n)` divides the current figure into `r` rows and `c` columns of equally spaced plot areas, and then establishes the `nth` of these (counting across the rows first) as the current figure. You do not have to draw in all of the areas you specify. Figure 11.1 was generated by the code shown in Listing 11.1.

In Listing 11.1:

Line 1: `close all` closes all figures currently open. This command should always be present at the beginning of a script but will be omitted from the example listings that follow.

Line 2: Specifies a suitable range of `x` values.

Line 3: Sets the first subplot region.

Line 4: This is the simple version of the `plot(...)` function introduced earlier, plotting `x` against `y` and automatically

Listing 11.1 Creating a subplot

```
 1. close all
 2. x = -2*pi:.05:2*pi;
 3. subplot(3,2,1)
 4. plot(x, sin(x))
 5. title('1 - sin(x)');
 6. subplot(3,2,2)
 7. plot(x, cos(x))
 8. title('2 - cos(x)');
 9. subplot(3,2,3)
10. plot(x, tan(x))
11. title('3 - tan(x)');
12. subplot(3,2,4)
13. plot(x, x.^2)
14. title('4 - x^2');
15. subplot(3,2,5)
16. plot(x, sqrt(x))
17. title('5 - sqrt(x)');
18. subplot(3,2,6)
19. plot(x, exp(x))
20. title('4 - e^x');
```

creating the axes, creating subplot 1, the plot in the top-left corner. Note that although in the figure seen here the line is gray, when you run the script, the line will appear in its default color, blue.

Line 5: The `title(...)` function puts the specified string at the top of the plot as its title.

Lines 6–8: Create subplot 2, the second plot on the first row.

Lines 9–11: Create subplot 3, the first plot on the second row.

Lines 12–14: Create subplot 4, the second plot on the second row.

Lines 15–17: Create subplot 5, the first plot on the bottom row.

Lines 18–20: Create subplot 6, the second plot on the bottom row.

Style Points 11.1

All of these capabilities are also available to the script that creates the plots, and you are very likely to want to generate a plot more than once. Therefore, it is unwise to put a significant amount of manual effort into adjusting a plot. It is better to experiment with the manual adjustments and then find out how to make the same adjustments in the script that creates the plots. This also leaves you a permanent record of how the plot was generated.

11.1.4 Manually Editing Plots

When a figure has been created, you are free to manipulate many of its characteristics by using its menu items and tool bars. They provide the ability to resize the plot, change the view characteristics, and annotate it with legends, axis labels, lines, and text callouts.

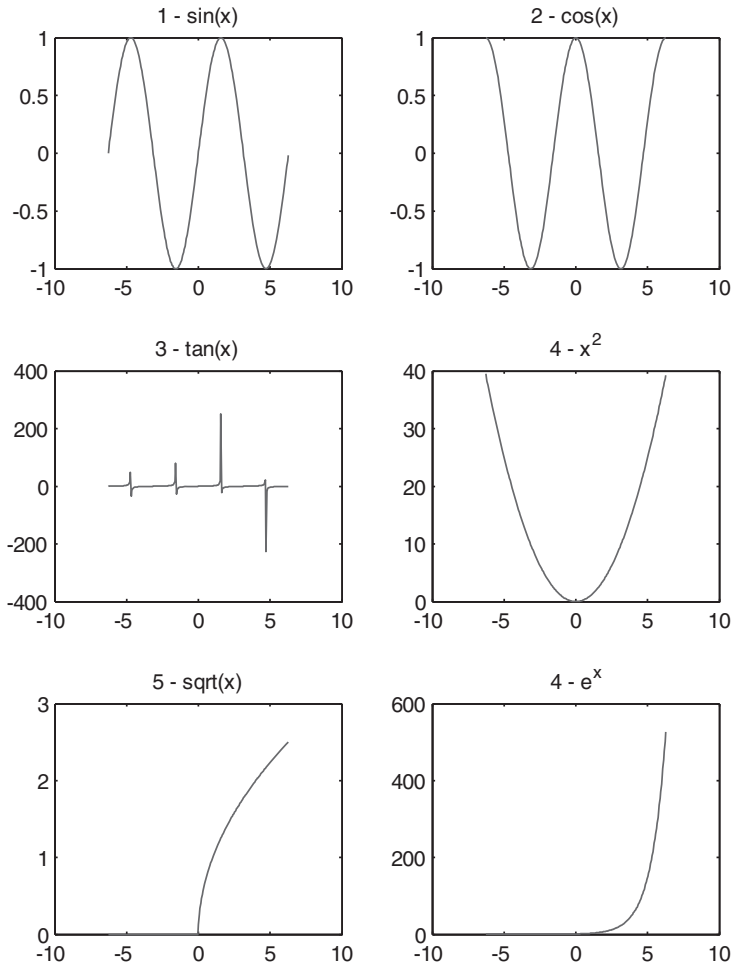


Figure 11.1 *Plotting effects*

11.2 2-D Plotting

11.2.1 Simple Plots

The basic function to use for 2-D plots is `plot(...)`. The normal use of this function is to give it three parameters, `plot(x, y, str)`, where `x` and `y` are vectors of the same length containing the x and y coordinates, respectively, and `str` is a string containing one or more optional line color and style control characters. A complete list of these control characters is included in Appendix A. If the vector `x` is omitted, MATLAB assumes that the x coordinates are 1:N, where N is the length of the `y` vector. If the `str` is omitted, the default line is solid blue. The MATLAB definition of this function also permits multiple `(x, y, str)` data sets in a single function call. It is always possible to produce the same result with multiple function calls in `hold on` mode.

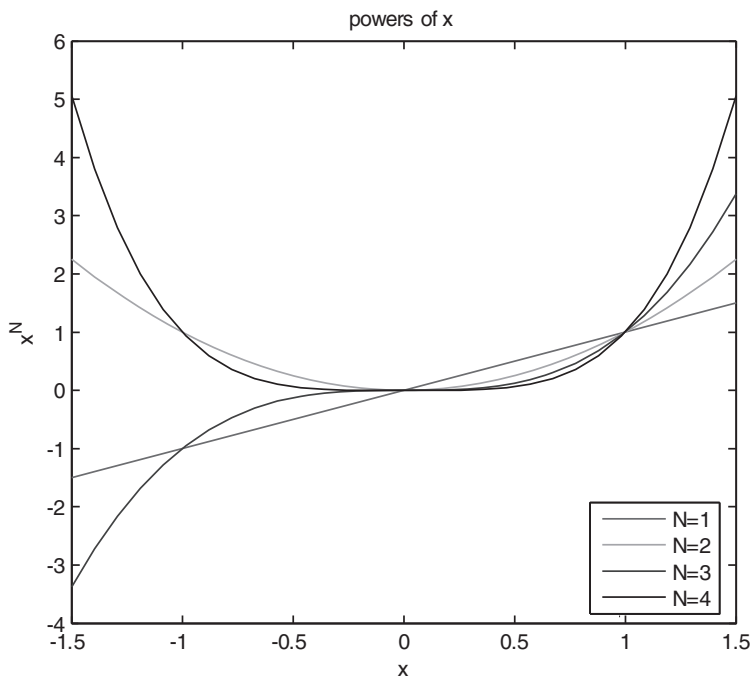


Figure 11.2 *Powers of x*

Listing 11.2 Simple 2-D plots

```
 1. x = linspace(-1.5, 1.5, 30);
 2. clr = 'rgbk';
 3. for pwr = 1:4
 4.     plot(x, x.^pwr, clr(pwr))
 5.     hold on
 6. end
 7. xlabel('x')
 8. ylabel('x^N')
 9. title('powers of x')
10. legend({'N=1', 'N=2', 'N=3', 'N=4'}, ...
11.           'Location','SouthEast')
```

Since we have already seen basic 2-D plotting at work, it should be sufficient to observe and comment on the simple example seen in Figure 11.2, generated by the code shown in Listing 11.2.

In Listing 11.2:

Line 1: Sets the range of x values.

Line 2: Color specifications for the plots—red, green, blue, and black.

Lines 4–7: Plot x, x^2, x^3, and x^4 with the above colors used in sequence.

Lines 7–11: Add enhancements to the plot as noted above.
Line 11: One of many possible parameters to the `legend(...)` function—this one forces its location to the lower-right corner of the figure, out of the way of the data.

11.2.2 Plot Options

In addition to the plot enhancement tools listed in Section 11.1.2, the following capabilities are available.

- Setting line styles and symbols to mark the data points (details in Appendix A)
- Using `plotyy(...)` to put a second axis on the right side of the figure
- Obtaining logarithmic plots on the x-axis (`semilogx(...)`), y-axis (`semilogy(...)`), or both axes (`loglog(...)`)

We strongly suggest that the reader experiment with these features and observe their effects.

11.2.3 Parametric Plots

Plotting is not restricted to the situation where the data along one axis are the independent variable and that along the other are dependent. Parametric plots allow the variables on each axis to be dependent on a separate, independent variable. That independent variable will define a path on the plotting surface. Consider the plot shown in Figure 11.3, which presents a simple exercise in transforming a circle into an airfoil. It was generated using the code shown in Listing 11.3.

Style Points 11.2

By convention, good engineers are expected to represent the data with appropriate line styles to avoid misleading the reader. For example, if you have some raw data that is only valid at the measurement points, it should be plotted with symbols only. Connecting the data with a line would imply that the data have some interpolated values, which may not be the case. On the other hand, if you calculate a theoretical curve that is good throughout the range of x, it should be plotted as a continuous curve, perhaps even at a better resolution (more x values) than the raw data samples.

Listing 11.3 Parametric plots

```
 1. th = linspace(0, 2*pi, 40);
 2. r = 1.1; g = .1;
 3. cx = sqrt(r^2-g^2) - 1; cy = g;
 4. x = r*cos(th) + cx;
 5. y = r*sin(th) + cy;
 6. plot( x, y,  'r' )
 7. axis equal
 8. grid on
 9. hold on
10. z = complex(x, y);
11. w = z + 1./z;
12. plot( real(w), imag(w), 'k' );
```

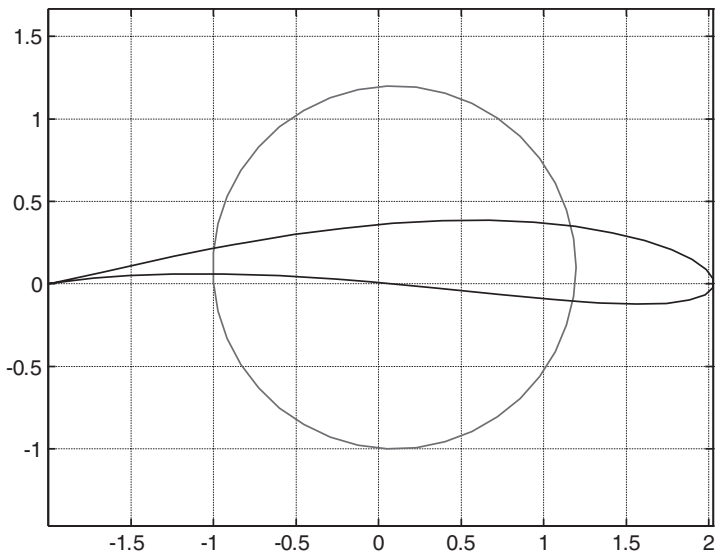


Figure 11.3 *Parametric 2-D plot*

In Listing 11.3:

Line 1: The independent variable in this case is the angle `th` varying from 0 to 2π.

Line 2: The particular transformation we use here requires a circle with a radius, `r`, slightly greater than 1 offset by a small distance, `g`, from the x-axis, passing through the point (−1, 0).

Line 3: We compute the center of the circle passing through the point (−1, 0).

Lines 4–5: A standard polar-to-Cartesian coordinate transformation computing the coordinates of the circle.

Line 6: Plots the two dependent variables `x` and `y` with a red line.

Line 7: Equalizes the axes and forces the circle to be drawn correctly.

Line 8: Displays a grid on which to estimate specific values.

Line 9: Here we want to add a second plot to the figure.

Lines 10–11: The Joukowski transformation is easiest when expressed in complex terms: if `z` is the path around the required circle, `w = z + 1/z` traces a very credible looking airfoil shape.

Line 12: Adds the plot of `w`, and reverts from the complex plane to plot the real and imaginary parts of the answer colored in black.

11.2.4 Other 2-D Plot Capabilities

You can also create some more exotic plots that are not necessary to understand the basic principles of plotting, but are powerful methods for visualizing real data:

- `bar(x, y)` produces a bar graph with the values in `y` positioned at the horizontal locations in `x`. The options available can be studied with `>> help bar`.
- `barh(x, y)` produces a bar graph with the values in `y` positioned at the horizontal locations in `x`. The options available can be studied with `>> help barh`.
- `fill(x,y,n)` produces a filled polygon defined by the coordinates in `x` and `y`. The fill color is specified by indexing `n` into the color map. The options available can be studied with `>> help fill`.
- `hist(y, x)` produces a histogram plot with the values in `y` counted into bins defined by `x`. The options available can be studied with `>>help hist`.
- `pie(y)` makes a pie chart of the values in `y`. For more options, see `>> help pie`.
- `polar(th, y)` makes a polar plot of the angle `th` (radians) with the radius `r` specified for each angle. For more options, see `>> help polar`.

11.3 3-D Plotting

Before attacking the details of plotting in three dimensions, it should be noted that even 2-D plots are actually 3-D plots. Consider the picture shown in Figure 11.4, which was generated originally as the 2-D plot in Figure 11.3. By selecting the Rotate 3-D icon on the tool bar and moving the mouse on your figure, it becomes apparent that what appeared to be a 2-D plot in the x-y plane is really a 3-D plot in the x-y-z plane "suspended in space" at z = 0.

11.3.1 Linear 3-D Plots

The simplest method of 3-D plotting is to extend our 2-D plots by adding a set of z values. In the same style as `plot(...)`, `plot3(x, y, z, str)` consumes three vectors of equal size and connects the points defined by those vectors in 3-D space. The optional `str` specifies the color and/or line style. If the `str` is omitted, the default line is solid blue.

Figure 11.5 shows three curves plotted in three dimensions, using the script shown in Listing 11.4. Each plot is in the z-x plane: the red curve at y = 0, the blue curve at y = 0.5, and the green curve at y = 1.

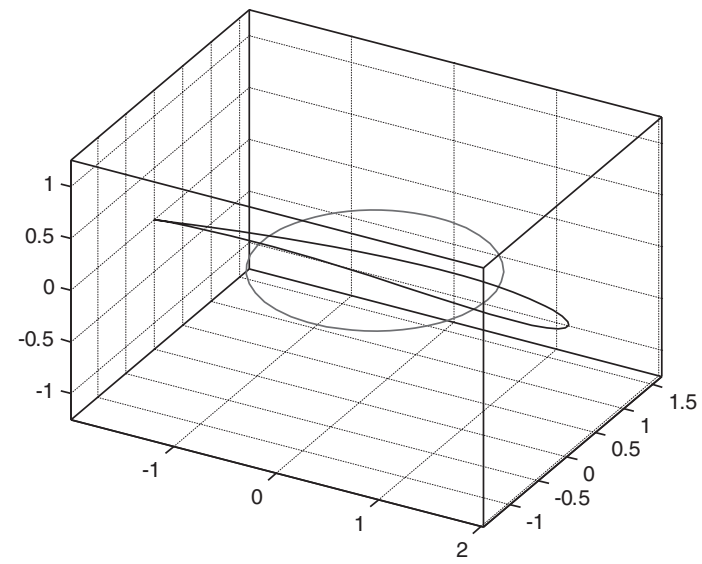


Figure 11.4 *Rotated 2-D plot*

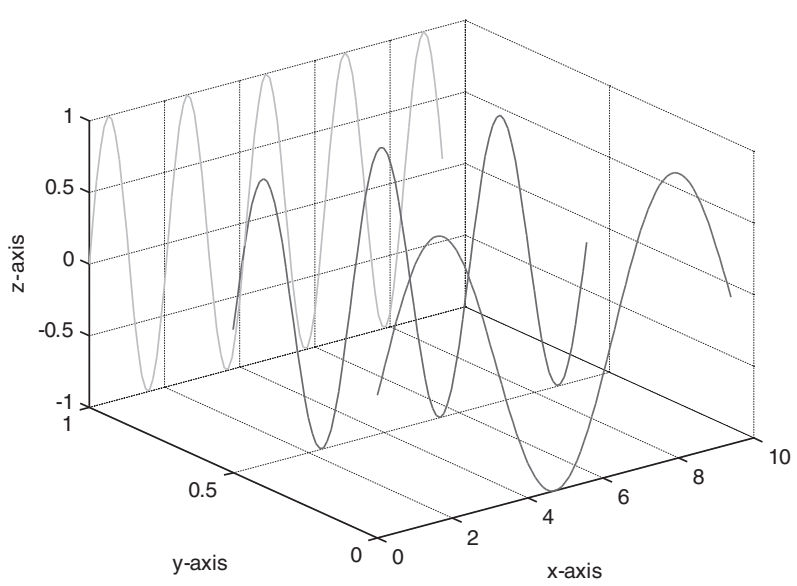


Figure 11.5 *3-D lines*

In Listing 11.4:

Line 1: Each plot has the same set of x values.
Lines 2–3: The y values for the first plot are all 0.
Lines 4–5: The second and third plots are sin(x) at different frequencies.

Listing 11.4 Simple 3-D line plots

```
 1. x=0:0.1:3.*pi;
 2. y1=zeros(size(x));
 3. z1=sin(x);
 4. z2=sin(2.*x);
 5. z3=sin(3.*x);
 6. y3=ones(size(x));
 7. y2=y3./2;
 8. plot3(x,y1,z1, 'r',x,y2,z2, 'b',x,y3,z3, 'g')
 9. grid on
10. xlabel('x-axis'), ylabel('y-axis'), zlabel('z-axis')
```

Lines 6–7: The y values of the second and third plots are all 0.5 and 1, respectively.

Lines 8–10: Plot and annotate the results.

11.3.2 Linear Parametric 3-D Plots

We can generalize the concept of parametric plots to 3-D, as shown in Figure 11.6, in which the x, y, and z values are mappings of some linear parameter. On the left side, the spiral is an example of a 3-D plot where two of the dimensions, x and y, are dependent on the third, independent parameter. The independent parameter in this example is the rotation angle, π, varying from 0 to 10π (five complete revolutions). The x and y

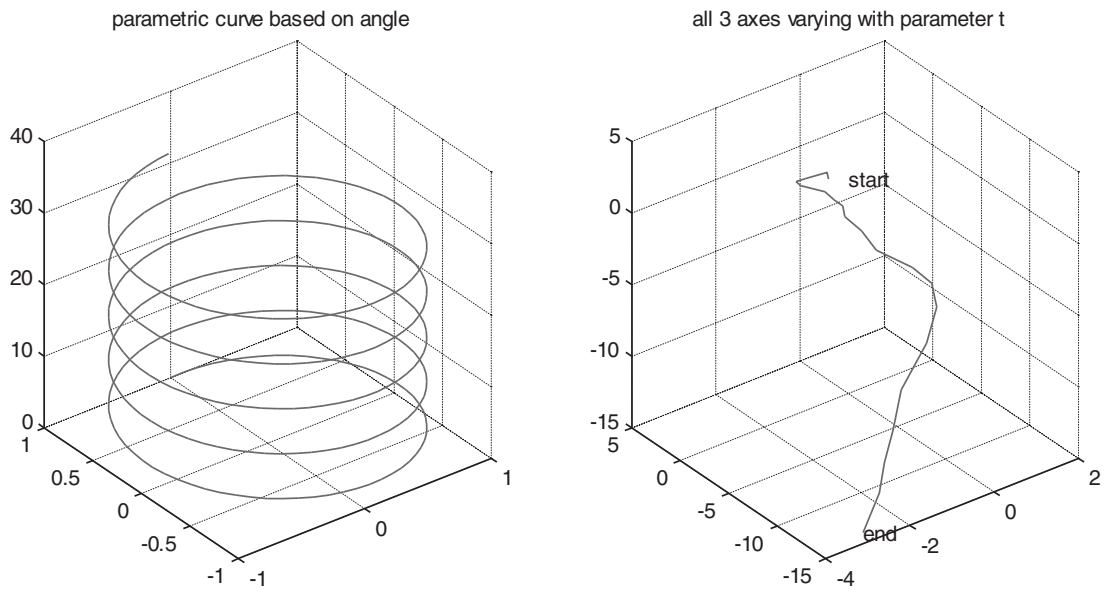


Figure 11.6 *Parametric 3-D plots*

values are mapped as sin(θ) and cos(θ)—the classic means of describing a circle. The spiral effect is accomplished by plotting θ on the z-axis.

The right half of Figure 11.6 illustrates a fully parametric plot, where the values of all three coordinates are mappings of an independent parameter, t. This particular example is a plot of the 3-D motion of a particle receiving random impulses in all three axes. Note the use of text anchored in x-y-z space to label points on the graph. The figure is drawn using Listing 11.5.

In Listing 11.5:

Lines 2–5: Draw the spiral plot with a simple `plot3(...)` call.

Lines 8–10: Define random velocity increments in x, y, and z.

Lines 11–13: Integrate to compute the position in x, y, z space. There will be a full discussion of integration in Chapter 15.

Lines 14–16: Plot and enhance the time history of the particle.

Lines 17 and 18: Add labels to indicate the start and end of the trace.

11.3.3 Other 3-D Plot Capabilities

If you are using MATLAB, you can also create some more exotic 3-D plots that are not necessary to understand the basic principles of plotting, but are powerful methods for visualizing real data:

- `bar3(x, y)` produces a bar graph with the values in `y` positioned at the horizontal locations in `x`. The options available can be studied with `>> help bar3`.

Listing 11.5 Linear parametric 3-D plots

```
 1. subplot(1, 2, 1)
 2. theta = 0:0.1:10.*pi;
 3. plot3(sin(theta),cos(theta),theta)
 4. title('parametric curve based on angle');
 5. grid on
 6. subplot(1, 2, 2)
 7. N = 20;
 8. dvx = rand(1, N) - 0.5   % random v changes
 9. dvy = rand(1, N) - 0.5
10. dvz = rand(1, N) - 0.5
11. x = cumsum(cumsum(dvx)); % integrate to get pos
12. y = cumsum(cumsum(dvy));
13. z = cumsum(cumsum(dvz));
14. plot3(x,y,z)
15. grid on
16. title('all 3 axes varying with parameter t')
17. text(0,0,0, 'start');
18. text(x(N),y(N),z(N), 'end');
```

- `barh3(x, y)` produces a bar graph with the values in `y` positioned at the horizontal locations in `x`. The options available can be studied with `>> help barh`.
- `pie3(y)` makes a 3-D pie chart of the values in `y`. For more options, see `>> help pie3`.

11.4 Surface Plots

In Section 11.3.2, we saw that data can be generated for all three axes based on one linear parameter. However, more dramatic graphics are produced by a different group of 3-D graphics functions that produce images based on mapping a 2-D surface. The underlying 2-D surface is sometimes referred to as *plaid* because of its conceptual similarity to a Scottish tartan pattern. To design such a pattern, one needs only to specify the color sequence of the horizontal and vertical threads. In the same way, we specify a plaid by defining vectors of the row and column data configurations.

11.4.1 Basic Capabilities

Three fundamental functions are used to create 3-D surface plots:

- `meshgrid(x, y)` accepts the x_{1*m} and y_{1*n} vectors that bound the edges of the plaid and replicates the rows and columns appropriately to produce xx_{n*m} and yy_{n*m}, containing the x and y values (respectively) of the complete plaid. This enables us in general to compute mappings for the 3-D coordinates of the figure we want to plot.
- `mesh(xx, yy, zz)` plots the surface as white facets outlined by colored lines. The line coloring uses one of many color maps (listed in Appendix A), where the color is selected in proportion to the `zz` parameter. You can turn the white facets transparent with the command `hidden off`.
- `surf(xx, yy, zz)` plots the surface as colored facets outlined by black lines. The line coloring by default is selected in proportion to the `zz` parameter. You can remove the lines by using one of a number of `shading` commands listed in Appendix A.

11.4.2 Simple Exercises

We will consider some simple situations that illustrate many of the features of surface drawing.

Drawing a Cube In the first example, in order to understand the underlying logic, we will develop the basic concept of drawing surfaces *without* the help of the `meshgrid(...)` function. Figure 11.7 shows the coordinates of a cube of side 2 units centered at the origin. Listing 11.6 shows the code that plots a cube from scratch. Figure 11.8 shows the results from this script. To

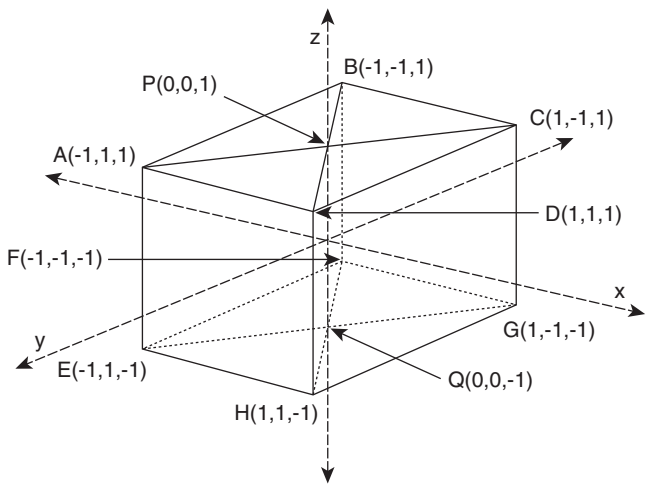


Figure 11.7 *A simple cube*

Figure 11.8 *The solid cube plot*

define the top and bottom of the cube, we must add the points P and Q. Although only one point each is required to define P and Q, the array must have the same number of columns in each row. Therefore, P and Q must be replicated five times to keep the arrays rectangular.

One could think about the way the `surf(...)` function works by drawing the line defined by the top row of the `xx`, `yy`, and `zz` arrays. Then it locates the line defined by the next row and makes a smooth surface between the two lines. Physically, this has the following effect:

- Beginning at point P, it draws expanding squares until it reaches ABCD

Listing 11.6 Simple solid cube

```
 1. xx = [  0  0  0  0  0  % P-P-P-P-P
 2.        -1 -1  1  1 -1  % A-B-C-D-A
 3.        -1 -1  1  1 -1  % E-F-G-H-E
 4.         0  0  0  0  0] % Q-Q-Q-Q-Q
 5. yy = [  0  0  0  0  0  % P-P-P-P-P
 6.         1 -1 -1  1  1  % A-B-C-D-A
 7.         1 -1 -1  1  1  % E-F-G-H-E
 8.         0  0  0  0  0] % Q-Q-Q-Q-Q
 9. zz = [  1  1  1  1  1  % P-P-P-P-P
10.         1  1  1  1  1  % A-B-C-D-A
11.        -1 -1 -1 -1 -1  % E-F-G-H-E
12.        -1 -1 -1 -1 -1] % Q-Q-Q-Q-Q
13. surf(xx, yy, zz)
14. colormap bone
15. axis equal
16. shading interp
17. view(-36, 44)
18. axis off
```

- "Sliding down" the sides of the cube to EFGH
- Shrinking that square down to the point Q

In Listing 11.6:

Lines 1–12: Establish the plaid defining the point P, the A-B-C-D plane, the E-F-G-H plane, and the point Q. Notice that the first corner is repeated on each row to close the figure shape.

Lines 13–18: Plot the cube top, sides, and bottom.

A Simple Parabolic Dish The simplest surface plots are obtained by defining a z value for each point on an x-y plaid. We will continue with a simple example illustrating the use of `meshgrid(...)` to define the plaid. Consider how we might plot the data shown in Figure 11.9. Before we look at the code, consider what the picture represents. Clearly, the independent variables are x and y, each covering the range from −3 to 3, each having seven discrete values. As the label indicates, the z values are calculated as the sum of x^2 and y^2. There are not, however, 14 z values as the range of x and y values might suggest, but 49! In order to plot the 3-D shape of our parabolic bowl, we must have a z value for every point on the x-y surface. Each of these points has a value of x corresponding to the reading on the x-axis, and a value of y from the y-axis. Therefore, the process of creating this plot has three parts:

1. Develop the underlying plaid specifying the x-y location of every point on the x-y plane.
2. Calculate the z values from the plaid.
3. Call a plotting function that will accept the plaid and these z values to produce the required plot.

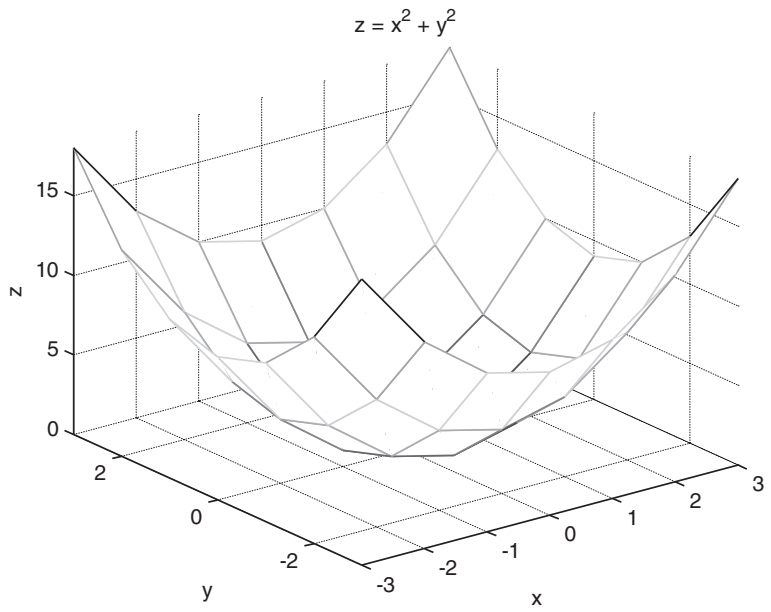


Figure 11.9 *A mesh plot*

The code to accomplish this is shown in Listing 11.7.

In Listing 11.7:

Line 1: The x and y vectors define the edges of the plaid.

Line 2: Generates the plaid.

Line 3: In this particular example, we map only the z coordinate, leaving the plaid (xx and yy) as the x and y coordinates of the figure.

Line 4: `mesh(...)` is one of the many functions that represent 3-D mappings of a plaid in different ways. Notice in the figure that the faces between line segments are solid white, and the line colors change with the z coordinate.

Lines 5–7: Annotate the plot.

Listing 11.7 Simple surface plot

```
1. x=-3:3; y = x ;
2. [xx,yy]=meshgrid(x,y);
3. zz=xx.^2 + yy.^2;
4. mesh(xx,yy,zz)
5. axis tight
6. title('z = x^2 + y^2')
7. xlabel('x'),ylabel('y'),zlabel('z')
```

Exercise 11.1 Exploring the simple plot

1. run script in Listing 11.7 without the semicolon on Line 2, and observe the following:

```
xx =
    -3  -2  -1   0   1   2   3
{etc}
yy =
    -3  -3  -3  -3  -3  -3  -3
-{etc}
     3   3   3   3   3   3   3
```

   Notice that in general, if `x` is length `m` and `y` is length `n`, the `xx` values consist of the `x` vector in rows replicated n times, and the `yy` values consist of the `y` vector as a column replicated m times. Together, they provide the underlying `x` and `y` values for the "floor" of the bowl plot from which the `z` values are computed to draw the picture.
2. Insert the line `hidden off` after `mesh(xx, yy, zz)`. Notice that the faces are now transparent.
3. Change `mesh(xx, yy, zz)` to `surf(xx, yy, zz)`. Notice that the panels are now colored and the lines are black. This form is also insensitive to the `hidden parameter`.
4. Replace `hidden off` with `shading flat`, and notice that the lines have disappeared.
5. Replace `shading flat` with `shading interp`, and notice that the surface coloring now varies smoothly.
6. Insert the line `colormap 'summer'` after `surf(xx, yy, zz)`. Look up `colormap` in Appendix A for details.
7. Do not forget to rotate your images and examine them from different points of view using the 3-D rotate tool bar icon.

Try Exercise 11.1 and make your observations.

Manipulating Plots Thoughtful students might develop their own tests to investigate the behavior of the following tools:

- The function `surfc(xx, yy, zz)` puts contour lines on the x-y plane base.
- The function `view(az, el)` changes the viewing angle. This is useful to capture a specific view angle after you have used the rotation tool to select a good presentation of the data.
- The command `colorbar` allows you to show how the colors are quantified on the plot.
- Adding a 4th parameter to `surf(xx, yy, zz, yy)` overrides the default color direction z with, in this case, the y direction.

- The 4th parameter can also be a function like `del2(zz)` that computes the second derivative, or curvature, of the plot, so now the coloring highlights the areas of maximum curvature.
- The 4th parameter can also be an image (see Chapter 13) that will appear to be pasted onto the plotting surface.
- For an eye-catching effect, add the line `lightangle(60, 45)` at the bottom of the script. This illuminates the surface with a light at the specified azimuth and elevation angle (degrees). The resulting faceted appearance can be alleviated by decreasing the granularity of the underlying plaid coordinates.

11.4.3 3-D Parametric Surfaces

Cylinder Consider first the construction of a cylinder as illustrated in Figure 11.10. One could consider this figure as a sheet of paper rolled up in a circular shape. We could visualize that piece of paper as a plaid of values, not of x-y in this case, but perhaps $x - \theta$. The range of x would be from 0 to the length of the cylinder, and the range of θ would be 0 to 360°.

To plot this, one would then merely need to create a plaid in x and θ, and then decide on the mapping from θ to the y and z values of the cylinder. The resulting picture is shown in Figure 11.11, and the code is shown in Listing 11.8.

In Listing 11.8:

Line 1: Constants to define the smoothness of the cylinder.

Lines 2–4: Define a plaid in x and θ. Note that only two points are needed in the x direction because that contour is straight.

Lines 5 and 6: The circular cross-section is achieved by using the parametric definition of a circle of a given radius.

Listing 11.8 Constructing a cylinder

```
 1. facets = 120; len = 2; radius = 1;
 2. thr = linspace(0, 2*pi, facets);
 3. xr = [0 len];
 4. [xx, tth] = meshgrid( xr, thr );
 5. yy = radius * cos(tth);
 6. zz = radius * sin(tth);
 7. surf(xx, yy, zz);
 8. shading interp
 9. colormap bone
10. axis equal,axis tight,axis off
11. lightangle(60, 45)
12. alpha(0.8)
13. view(-20, 35)
```

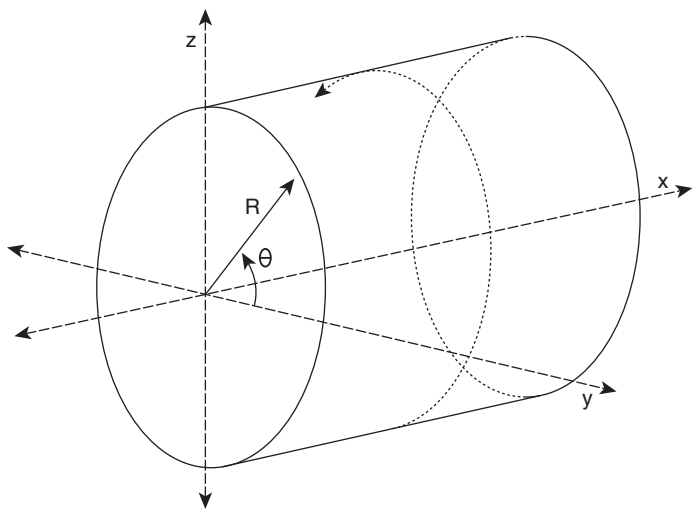


Figure 11.10 *Creating a cylinder image*

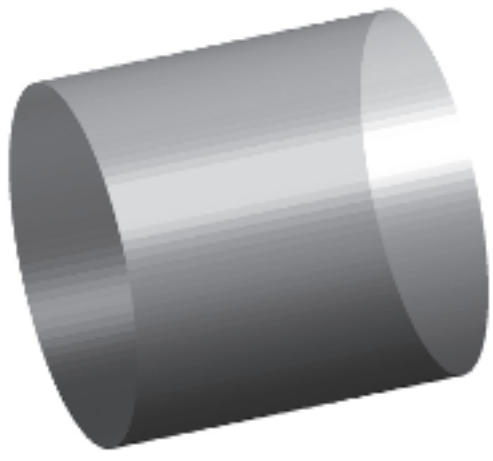


Figure 11.11 *A cylinder plot*

Line 9: Changes the color to a pleasant metallic scale.
Line 10: Squares up and removes the axes.
Line 11: Illuminates the figure.
Line 12: Sets the transparency of the surface so that a portion of the hidden details can show through.

Sphere Now, we construct a sphere as shown in Figure 11.12, starting with the cylinder. However, instead of using a constant radius in the x direction, we will calculate the radius in that direction by rotating a second angle, ψ, from 0 to 180°. Think of this as mapping or "wrapping" a plaid with two angles as the independent variables around the sphere. The coordinate in the x direction would be r cos ψ, and the radii of the y-z

Figure 11.12 *A sphere*

Listing 11.9 Constructing a sphere

```
 1. facets = 120; radius = 1;
 2. thr = linspace(0, 2*pi, facets); % range of theta
 3. phir = linspace(0, pi, facets); % range of phi
 4. [th, phi] = meshgrid( thr, phir );
 5. x = radius * cos(phi);
 6. y = radius * sin(phi) .* cos(th);
 7. z = radius * sin(phi) .* sin(th);
 8. surf(x, y, z);
 9. shading interp
10. colormap copper
11. axis equal, axis tight, axis off
12. lightangle(60, 45)
```

circles would be r sin ψ. The code for drawing this sphere is shown in Listing 11.9.

In Listing 11.9:

Line 1: The `radius` set here is the sphere radius.
Lines 2 and 3: Set the ranges of θ and ψ.
Line 4: Builds the plaid in θ and ψ.
Line 5: As ψ rotates, the value of `x` varies as its cosine.
Lines 6 and 7: The radius of rotation about the x-axis varies as the sine of ψ.
Lines 8–12: Draw and annotate the plot.

11.4.4 Bodies of Rotation

The cylinder and sphere drawn in the above section are special cases of a more general form of solid body. Bodies of rotation are created in general

by rotating a general function `v = f(u)` defined over a range of `u` values about the x or z axes. Note: this is perfectly general because rotating such a function about the y-axis would result merely in "smearing" the function across a flat surface in the x-z plane. We use z rather than y for the dependent variable here because in our 3-D plots, the z-axis is drawn as the vertical axis. In general, we make no claims about the nature of `f()`. It could be a rational function, or merely a "lookup table" specifying a value of `v` for every `u`.

Rotating Continuous Functions First, we consider rotating a continuous function `v = f(u)` about the x and z axes.

- To rotate `v = f(u)` about the x-axis, we could consider this equation as `r = f(x)`. Figure 11.13 shows the logic of this rotation. The independent variable is x, and the values of y and z are computed as the usual polar-to-Cartesian conversion:

  ```
  y = r cos(θ)
  z = r sin(θ)
  ```

  Notice that these are the two axes about which we are not rotating.
- To rotate `v = f(u)` about the z-axis, we could consider this equation as `z = f(r)`. Figure 11.14 shows the logic of this rotation. The independent variable is now `r`, and the values of x and y are computed as the usual polar-to-Cartesian conversion:

  ```
  x = r cos(θ)
  y = r sin(θ)
  ```

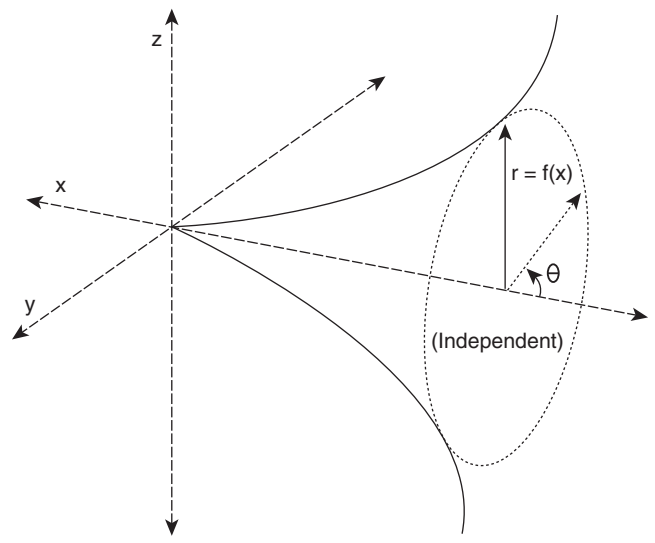


Figure 11.13 *Rotating* `v = f(u)` *about the x-axis*

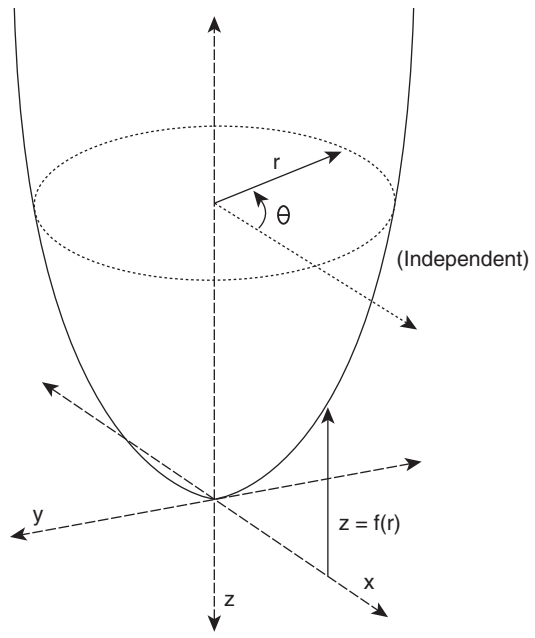


Figure 11.14 *Rotating* `v = f(u)` *about the z-axis*

Notice again that these are the two axes about which we are not rotating. Notice also a simple rule of thumb: if you rewrite `v = f(u)` correctly for each rotation, the independent variable is always the parameter of `f(...)`.

Figure 11.15 shows the result of the rotations generated by the code shown in Listing 11.10.

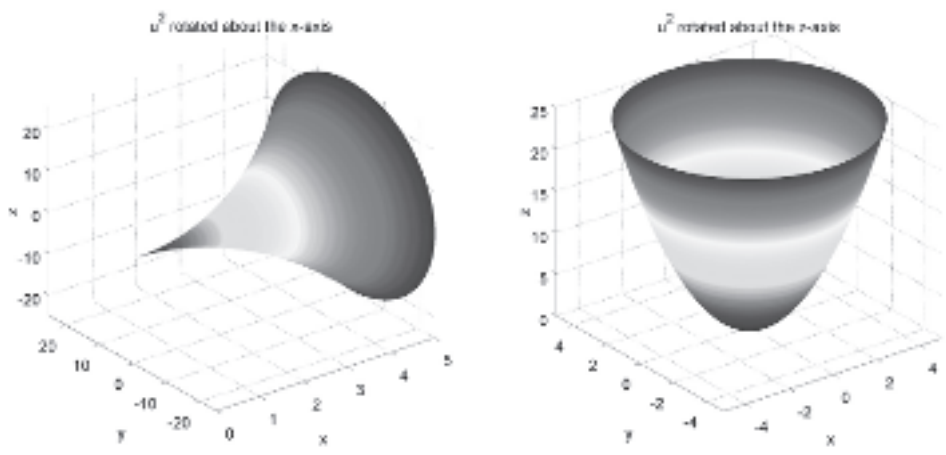


Figure 11.15 *Rotation of* u^2

Listing 11.10 Rotating $v = u^2$ about the x and z axes

```
 1. facets = 100;
 2. u = linspace(0, 5, facets);
 3. th = linspace(0, 2*pi, facets);
 4. [uu tth] = meshgrid(u, th);
    % rotate about the x-axis
 5. subplot(1, 2, 1)
 6. rr = uu.^2;
 7. xx = uu;
 8. yy = rr .* cos(tth);
 9. zz = rr .* sin(tth);
10. surf(xx, yy, zz, xx);
11. shading interp, axis tight
12. xlabel('x'), ylabel('y'), zlabel('z')
13. title('u^2 rotated about the x-axis')
    % rotate about the z-axis
14. subplot(1, 2, 2)
15. rr = uu;
16. zz = rr.^2;
17. xx = rr .* cos(tth);
18. yy = rr .* sin(tth);
19. surf(xx, yy, zz);
20. shading interp, axis tight
21. xlabel('x'), ylabel('y'), zlabel('z')
22. title('u^2 rotated about the z-axis')
```

In Listing 11.10:

Lines 1–4: Set up the plaid of `u`, the independent variable for the function, and θ for the rotations.

Lines 6–13: Compute the rotation about the x-axis. Notice that when rotating about a specific axis, that axis must be treated separately; the other two axes will always have the form of a polar-to-Cartesian transformation. In rotating about the x-axis, since u is the independent variable for our function, we only need to compute the yy and zz values.

Line 10: We use the fourth parameter to `surf(...)` to set the direction of color variation.

Lines 15–22: Compute the z-axis rotation. Some apparent sleight of hand is necessary here. In this case, the axis containing the independent variable is being rotated about the z-axis. Because the radius of the rotated surface is the original independent variable, `uu`, we copy `uu` to the variable radius. Then we define `xx` together with `yy` as the polar-to-Cartesian transformation to achieve the rotation. In this case, the z value of the surface is `f(u)`, u^2.

Rotating Discrete Functions There is no need to restrict ourselves to continuous functions as the profiles for bodies of rotation. Figure 11.16 shows the 2-D profile of a fictitious machine part and the picture created

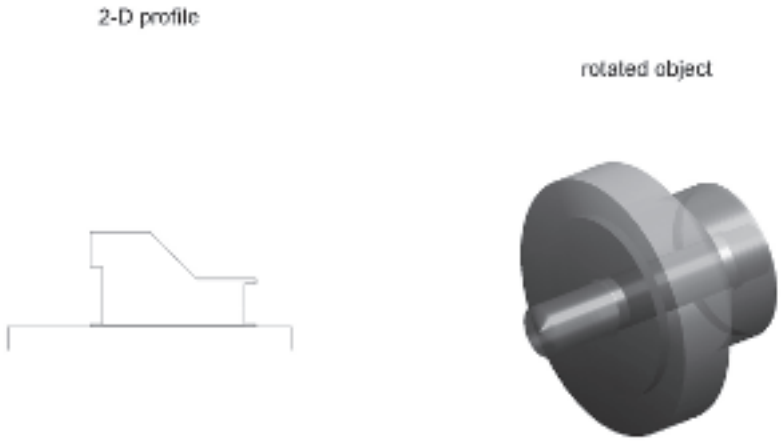


Figure 11.16 *Rotation of an irregular shape*

when that profile is rotated about the x-axis. The figure was generated by the code shown in Listing 11.11.

In Listing 11.11:

Lines 1–9: Define and plot the initial 2-D profile.

Lines 10–22: Perform the rotation about the x-axis. The only unusual idea here is how to turn this discrete collection of points

Listing 11.11 Rotating an irregular shape

```
 1. u = [0 0 3 3 1.75 1.75 2 2 1.75 1.75 3 4 ...
 2.      5.25 5.25 5 5 5.25 5.25 3 3 6 6];
 3. v = [0 .5 .5 .502 .502 .55 .55 1.75 1.75 ...
 4.       2.5 2.5 1.5 1.5 1.4 1.4 ...
 5.      .55 .55 .502 .502 .5 .5 0];
 6. subplot(1, 2, 1)
 7. plot(u, v, 'k')
 8. axis ([-1 7 -1 3]), axis equal, axis off
 9. title('2-D profile')
10. facets = 200;
11. subplot(1, 2, 2)
12. [xx tth] = meshgrid( u, linspace(0, 2*pi, facets) );
13. rr = meshgrid( v, 1:facets);
14. yy = rr .* cos(tth);
15. zz = rr .* sin(tth);
16. surf(xx, yy, zz);
17. shading interp
18. axis square, axis tight, axis off
19. colormap bone
20. lightangle(60, 45)
21. alpha(0.8)
22. title('rotated object')
```

into the equivalent of `v = f(u)`. Line 12 shows an elegant way to solve this dilemma. After going through the `meshgrid(..)` to produce a plaid of `xx` and `tth`, we run `meshgrid(...)` again, but keeping only the first result, `rr`.

Rotating about an Arbitrary Axis Bodies of rotation are not confined to rotating about the x, y, or z axes. The simplest approach to rotating `z = f(x)` about an arbitrary axis is as follows:

- Calculate the matrix that will place your axis of rotation along the x-axis (see Chapter 12)
- Transform u and v with that rotation
- Rotate the transformed u and v about the x-axis
- Invert the transformation on the resulting surface

11.4.5 Other 3-D Surface Plot Capabilities

The MATLAB language also defines special-purpose functions to enhance the quality of surface plots:

- `alpha(x)` sets the transparency of the surfaces. `0<=x<=1`, where `0` means completely transparent and `1 (the default value)` is opaque. The options available can be studied with `>> help alpha`.
- `contour(z)` produces a contour plot of the plaid surface defined by `z`. The options available can be studied with `>> help bar3`.
- `[x,y,z] = cylinder(n)` constructs the `meshgrid` for a cylinder with `n` facets in each direction. For more options, see `>> help cylinder`.
- `[x,y,z] = ellipsoid(n)` constructs the `meshgrid` for an ellipsoid with `n` facets in each direction. For more options, see `>> help ellipsoid`.
- `[x,y,z] = sphere(n)` constructs the `meshgrid` for ansphere with `n` facets in each direction. For more options, see `>> help sphere`.
- `meshc(x,y,z)` makes a mesh plot with contours below. For more options, see `>> help meshc`.
- `meshz(x,y,z)` makes a mesh plot with vertical line extensions. For more options, see `>> help meshz`.
- `surfc(x,y,z)` makes a surface plot with contours below. For more options, see `>> help surfc`.
- `surfz(x,y,z)` makes a surface plot with vertical line extensions. For more options, see `>> help surfz`.
- `waterfall(x,y,z)` makes a mesh plot with vertical line extensions only in the x direction. For more options, see `>> help waterfall`.

Figure 11.17 *The Klein bottle*

11.4.6 Assembling Compound Surfaces

We can assemble more complex solid bodies by constructing simple surfaces and concatenating the data before submitting it to the rendering machine. Shapes of considerable complexity can be assembled this way. Consider, for example, the Klein bottle, a well-documented example of topological curiosity. The particular example shown in Figure 11.17 was constructed by building the individual components and then concatenating the arrays.

The code is a little too complex to be included here, but can be found on the companion Web site.

11.5 Manipulating Plotted Data

Two new features introduced with MATLAB 7.6 (R2008a) allow you to interact with the data presented in a plot. Brushing allows you to select portions of the data presented in a plot and make changes to the values presented. Linking allows you to connect the plotted data to the underlying data source, so that when you make changes to the plotted data, these changes are reflected in the data source. Whereas these tools allow the user to change the appearance of data presentations interactively, a careful user would return to the original tools that created the plots and explicitly insert the logic that changes the appearance of the results. This provides a traceable set of programs that show exactly how the data were generated.

11.6 Engineering Example—Visualizing Geographic Data

You have been given two files of data: `atlanta.txt`, which presents the streets of Atlanta in graphical form, and `ttimes.txt`, which gives the travel times between Atlanta suburbs and the city center. You have been asked to

present these data sets in a manner that will help to visualize and validate the data.

11.6.1 Analyzing the Data

First, we proceed to determine the nature of the data by opening the files and examining their format and content.

1. Determine the file format: the first step is to open the data files in a plain text editor. The format appears to be consistent with that of a text file delimited by tab characters. Since there are no strings in the file, it should be suitable to be read using the built-in `dlmread(...)` function.
2. Discern the street map file content: Table 11.1 shows the first few lines of the file `atlanta.txt` simplified by omitting certain irrelevant columns. The numbers in columns 3–6 are pairs, the first of the pair being a large negative number, and the second a smaller positive number. Assuming that each row of this file is a street segment, these could be the x-y coordinates of the ends of a line. A little thought confirms this guess when we realize that the latitude of Atlanta is –84° 42′ relative to the Greenwich meridian, and its longitude is 33° 65′—clearly, the values in these columns are 1,000,000 times the latitude and longitude of points within the city, probably each end of street segments. Column 7 contains numbers mostly in the range 1–6, which could indicate the type of street. We could explore this idea by coloring each line according to that value.
3. Discern the travel time file content: Table 11.2 shows the first few lines of the file `ttimes.txt` simplified by omitting certain irrelevant columns. The

Table 11.1 Street map data

...	...	–84546100.00	33988160.00	–84556050.00	33993620.00	1.00	...
...	...	–84546080.00	33988480.00	–84558400.00	33995480.00	1.00	...
...	...	–84243880.00	33780010.00	–84249980.00	33800840.00	1.00	...
{etc}							

Table 11.2 Travel time data

1	1	...	–84575725	33554573	14.34
1	2	...	–84569612	33554573	0
1	3	...	–84563499	33554573	0
1	4	...	–84557387	33554573	0
{ etc}					

same latitude/longitude values occur in columns 4 and 5, but they are not repeated, suggesting that the data in this file are in a different form. Examining the first two columns, the numbers in column 2 cycle repeatedly from 1 to 75, with column 1 counting the number of cycles up to 75. Furthermore, the values in column 5 are the same whenever column 1 is the same, and the values in column 4 are the same whenever the value in column 2 matches. This seems to be much like the plaid that results from a `meshgrid(...)` function call. The values in column 6 then become evident—they would be the z values of the plaid, and it seems reasonable to assume that they represent the travel time in minutes.

11.6.2 Displaying the Data

With this much understanding of the data sources, we proceed to solve the problem of presenting the data. The script shown in Listing 11.12 shows the code used to visualize these data files.

Listing 11.12 Map data analysis

```
    % draw the streets
 1. raw = dlmread('atlanta.txt');
 2. streets = raw(:,3:7);
 3. [rows,cols] = size(streets)
 4. colors = 'rgbkcmo';
 5. for in = 1:rows
 6.      x = streets(in,[1 3])/1000000;
 7.      y = streets(in,[2 4])/1000000;
 8.      col = streets(in,5);
 9.      col(col < 1) = 7;
10.      col(col > 6) = 7;
11.      plot(x,y,colors(col));
12.      hold on
13. end
    % plot the travel times
14. tt = dlmread('ttimes.txt');
15. [rows,cols] = size(tt)
16. for in = 1:rows
17.      r = tt(in, 1); c = tt(in, 2);
18.      xc(r,c) = tt(in, 4)/1000000;
19.      yc(r,c) = tt(in, 5)/1000000;
20.      zc(r,c) = tt(in, 6);
21. end
22. surf(xc, yc, zc)
23. shading interp
24. alpha(.5)
25. grid on
26. axis tight
27. xlabel('Longitude')
28. ylabel('Latitude')
29. zlabel('Travel Time (min)')
30. view(-30, 45)
```

In Listing 11.12:

Line 1: Reads the street map data.
Lines 2–3: Extract the relevant columns and determine the size of the array.
Line 4: Color symbols to use for the lines.
Line 5: Traverses the rows of the file.
Lines 6 and 7: Extract the longitude and latitude in degrees.
Lines 8–10: Extract and limit the line colors.
Lines 11 and 12: Plot the street lines on the same figure.
Lines 14 and 15: Read the travel times.
Line 16: Constructs the plaid by traversing the array.
Line 17: Extracts the row and column numbers.
Lines 18–20: Extract the plaid values.
Lines 22–30: Plot and display the results.

Figure 11.18 shows the resulting plot. As a credibility check, the plot has been rotated to look straight down on the map. Rotate the plot to other view angles to understand the 3-D nature of the information. The travel time surface shows "valleys" of low travel times that follow the paths of the major expressways through the city.

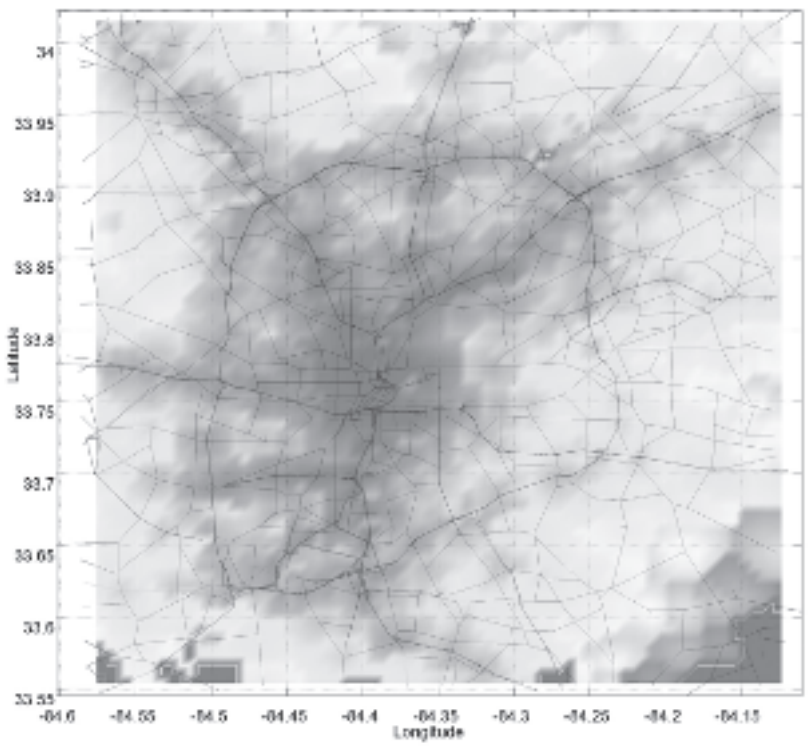


Figure 11.18 *Atlanta travel times*

Chapter Summary

This chapter presented the principles and practice of plotting:

- Basic 2-D line plots are accomplished by using `plot(x,y)`, where `x` is the independent variable and `y` the dependent variable
- 2-D parametric plots are accomplished by using `plot(x,y)`, where both `x` and `y` are dependent on another independent variable
- 3-D line and parametric plots are accomplished by using `plot3(x,y,z)`
- Basic 3-D surface plots are accomplished by building a plaid using `[xx yy] = meshgrid(x,y)`, computing the `zz` layer as a function of `xx` and `yy`, and then plotting the surface using `mesh(xx, yy, zz)` or `surf(xx, yy, zz)`
- Parametric surface plots, like parametric line plots, are achieved by building the plaid with two independent variables and making `xx`, `yy`, and `zz` functions of those independent variables
- Bodies of rotation are a special case of parametric surface plots where one of the independent variables is an angle with values between 0 and 2π.

Special Characters, Reserved Words, and Functions—2-D

Special Characters, Reserved Words, and Functions	Description	Discussed in This Section
`axis(...)`	Freezes the current axis scaling for subsequent plots or specifies the axis dimensions	11.1.2
`bar`	Generates a bar graph	11.2.4
`barh`	Generates a horizontal bar graph	11.2.4
`clf`	Clears the current figure	11.1.1
`close all`	Closes all graphics windows	11.1.1
`colormap <spec>`	Specifies a sequence of colors to be used when a cycle of color values is required	11.1.2
`figure`	Opens a new figure window	11.1.1
`fill(x,y,n)`	Fills a polygon defined by `x` and `y` with color index `n`	11.2.4
`grid off`	Turns the grid off (default is on)	11.1.2
`grid on`	Adds a grid to the current and all subsequent graphs in the current figure	11.1.2
`hist`	Generates a histogram	11.2.4
`hold off`	Sets a flag to erase figure contents before adding new information (the default state)	11.1.2

Special Characters, Reserved Words, and Functions	Description	Discussed in This Section
`hold on`	Sets a flag not to erase figure contents before adding new information	11.1.2
`legend(ca)`	Adds a legend to a graph	11.1.2
`loglog`	Generates an x-y plot, with both axes scaled logarithmically	11.2.4
`pie`	Generates a pie chart	11.2.4
`plot(...)`	Creates an x-y plot	11.1.2
`polar`	Creates a polar plot	11.2.4
`semilogx`	Generates an x-y plot, with the x-axis scaled logarithmically	11.2.4
`semilogy`	Generates an x-y plot, with the y-axis scaled logarithmically	11.2.4
`shading <spec>`	Shades a surface according to the specification	11.1.2
`subplot(plts, n)`	Divides the graphics window into sections available for plotting	11.1.1
`text(x,y,{z,} str)`	Adds a text string to a graph	11.1.2
`title(str)`	Adds a title to a plot	11.1.2
`view(az,el)`	Sets the angle from which to view a plot	11.1.2
`xlabel(str)`	Adds a label to the x-axis	11.1.2
`ylabel(str)`	Adds a label to the y-axis	11.1.2
`zlabel(str)`	Adds a label to the z-axis	11.1.2

Special Characters, Reserved Words, and Functions—3-D

Special Characters, Reserved Words, and Functions	Description	Discussed in This Section
`alpha(x)`	Sets the transparency of the surface	11.3.3
`bar3`	Generates a 3-D bar graph	11.3.3
`barh3`	Generates a horizontal 3-D bar graph	11.3.3
`contour(xx, yy, zz)`	Generates a contour plot	11.4.5
`cylinder(n)`	Constructs the plaid for a cylinder with `n` facets	11.4.5
`ellipsoid(n)`	Constructs the plaid for an ellipsoid with `n` facets	11.4.5
`lightangle(az,el)`	Sets the angle of a light source, angles in degrees	11.4.5
`mesh(xx,yy,zz)`	Generates a mesh plot of a surface	11.4.1

Special Characters, Reserved Words, and Functions	Description	Discussed in This Section
`meshc(xx,yy,zz)`	Generates a mesh plot of a surface with a contour below it	11.4.5
`meshz(xx,yy,zz)`	Generates a mesh plot of a surface with vertical line extensions	11.4.5
`[rr cc] = meshgrid(r,c)`	Creates a plaid for 3-D plots	11.4.1
`pie3`	Generates a 3-D pie chart	11.3.3
`plot3(...)`	Generates a 3-D line plot	11.3.1
`sphere(n)`	Example function used to demonstrate graphing	11.4.5
`surf(xx,yy,zz)`	Generates a surface plot	11.4.1
`surfc(xx,yy,zz)`	Generates a combination surface and contour plot	11.4.5
`waterfall(xx,yy,zz)`	Generates a mesh plot of a surface with vertical line extensions in the x direction only	11.4.5

Self Test

Use the following questions to check your understanding of the material in this chapter:

True or False

1. The `plot(...)` function needs only one parameter to function correctly.
2. Plot enhancement functions may be called before or after the function that plots the data.
3. You must provide plots for all the specified sub-plot areas.
4. `meshgrid(...)` accepts vectors of length m and n that bound the edges of the plaid and produces two arrays sized m × n giving the complete plaid.
5. To construct a parametric surface, both independent parameters must be angles.
6. When rotating a function about the y-axis, the variables along the x and y axes are computed from a classic polar-to-Cartesian conversion.
7. To compute a body of rotation, the curve must be a continuous, differentiable function.
8. Bodies of rotation are confined rotating about the x, y, or z axes.

Fill in the Blanks

1. Each time `figure` is called, a(n) _____ is made available, with figure number _____.
2. To prepare for plotting, put _____ or _____ at the beginning of your script.
3. Parametric plots allow the variables on each axis to be _____ on a(n) _____, _____ variable.
4. The simplest surface plots are obtained by defining a(n)_____ value for each point on _____.
5. We construct a sphere by wrapping a(n) _____with two _____ as the independent variables around the sphere.
6. Bodies of rotation are created by rotating a(n) _____ about a(n) _____.

Programming Projects

1. Write a script that creates six sub-plots in two columns each with three rows. Each plot should have an appropriate title and labels on the `x` and `y` axes. The plot in the top left sub-plot should be `y = cos(θ)` for values of θ from -2π to 2π. Subsequent plots going across the rows before going down the columns should be of `y = cos(2θ), y = cos(3θ), etc., to y = cos(6θ)` over the same range of θ.
2. Your task is to create a script called `thisPlot`. This script should do the following:
   a. Ask the user to enter in a positive number, N, greater than 5.
   b. Calculate the factorial for each number from 1 to N. Each of these values should be stored into a vector.
   c. Display a graph titled `'Logarithmic Growth'`, where the logarithms for each of the factorials are displayed.
   d. Add to the graph a continuous linear line that follows the equation `y = x` with `x` values from `1` to `N`.
   e. Since the numbers will have different magnitudes, use `plotyy` to plot the linear values on the right hand axis.
3. Write a function called `sineGraph` that graphs a sine function four times between the interval `[start,stop]` on the same graph. The `start` and `stop` values should be parameters of the function. The number of points per interval will vary. More specifically:
   - The first time you graph the sine function, you should have two evenly spaced points, `start` and `stop`
   - The next plot should have four evenly spaced points—`start`, `stop`, and two points between them

- The third should have eight evenly spaced points and the fourth 256 points.
- Make sure to add a legend and a title—`'Multiple graphs on one plot'`—and to label the axes. Make sure that each line has a different color.
- The function should return the `x` and `y` values for the 256 point set.

Test your function with the following intervals `[0,π/2]`, `[0,2π]`, `[0,4π]`, `[0,16π]`

4. This programming problem will compare the `surf(...)` and `mesh(...)` functions by putting two 3-D side-by-side plots for comparison using `subplot(...)`. You should label all axes accordingly (`'X-axis'`, `'Y-axis'`, etc.) and title your plot corresponding to the problem statement.
   a. On the left side, plot the function `f(x,y)=x^2*cos(y)` in the range `x= -5:5 and y= -5:5` using mesh and name your plot `'Using Mesh'`.
   b. On the right side, plot the same function, in the same range, but using `surf`. Name your plot `'Using Surf'`.
5. Georgia Tech wants to tear down the Campanile and build a new one that is ridiculously tall. However, before it is built, it needs you to model it. Using the equation `z = 1/(x^2 + y^2)` as the model, write a script that will plot the Campanile. Your domain should be `-.75 <= x <=.75` and `-.75 <= y <= .75` using an increment of .05 for each range. Set your axes such that all of the x, y domain is seen and z runs from 0 to 300. Use `surf(...)` to plot your image.
6. You are provided the file `'data.csv'`, which contains two columns of numbers. Each column contains the same number of elements. The first row contains the titles of the x and y values, respectively. Create a script called `spreadSheetPlot` that plots the data in this file. The first column represents your `x` values, and the second column is your `y` values. Read the numbers from the file and make a plot of the `x` vs. `y` values. Title your plot `'spreadSheetPlot'` and use the first row data to label the `x` and `y` axes. For example, the spreadsheet might look like:

	A	B
1	time(sec)	distance(ft)
2	0	0.84
3	0.2	0.23
4	0.4	0.77
5	0.6	1.06
6	0.8	1.28
7	1	1.48
8	1.2	1.63
9	...	...
10	...	...
11	...	...
12		

7. You just realized that February 14th has passed and you haven't gotten anything for your Valentine. Since your date is a CS major, sending the lucky person a coded heart seems like a cool and sincere thing to do. Make sure that you follow each and every instruction carefully, or your heart will end up broken. Trust us.

   You are going to write a script to draw this heart using the following steps:

   a. Create a plaid [xx, yy] using x values with range (0 to 2π, with an interval 0.05π) and y values with the range (0 to 1, with an interval 0.05).
   b. Define the following variables:

      ```
      c=[0.1 + 0.9*(π-abs(xx - π))/π ] .* yy
      aa = c .* cos(xx)
      bb = c.* sin(xx)
      zz = (-2)*aa.^3 + (3/2)*c.^2 + 0.5
      ```

   c. Plot `zz` against `aa` and `bb` using the `surf()` function with interpolated colors.

8. Write a function named `plotRotation` that takes in two vectors, `x` and `z`, and a vector `th`. Your function should plot three plots in the same figure by using the `subplot` command. The figure should have `1×3` plots. The plots should be as follows:

   a. z vs. x, titled `'z vs. x'`. Note that you will have to use `plot3()` to correctly plot this in the x-z plane rather than the x-y plane a `plot()` would do. Also, you should use `view(0, 0)` to make the plot produced by `plot3()` show up as 2-D.
   b. z vs. x rotated around the x-axis using `mesh()` with flat shading and a square axis, titled `'z vs. x about x using mesh'`.
   c. z vs. x rotated around the z-axis using `surf()` with `interp` shading and a square axis, titled `'z vs. x about z using surf'`.

   For plots b and c, the input vector `th` should be used for your independent vector theta, which is used to convert from polar-to-Cartesian coordinates. Don't forget to title and label each of the three plots.