3 Plotting In Matlab

In this chapter we introduce Matlab technique to draw the graph of functions in a variety of formats. We will begin our work in the plane, plotting the graphs of function, then moving to graphs defined by parametric and polar equations. We'll then move to 3-space and investigate the nature of curves and surfaces in space.

Table of Contents

Copyright

All parts of this Matlab Programming textbook are copyrighted in the name of Department of Mathematics, College of the Redwoods. They are not in the public domain. However, they are being made available free for use in educational institutions. This offer does not extend to any application that is made for profit. Users who have such applications in mind should contact David Arnold at david-arnold@redwoods.edu or Bruce Wagner at bruce-wagner@redwoods.edu.

This work (including all text, Portable Document Format files, and any other original works), except where otherwise noted, is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 2.5 License, and is copyrighted ©2006, Department of Mathematics, College of the Redwoods. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/2.5/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

3.1 Plotting in the Plane

In the last section we investigated the various array operations available in Matlab. We discovered that most Matlab functions are "array smart," operating on a vector or matrix with the same ease as they do on single numbers. For example, we can take the square root of a single number.

```
>> sqrt(9)
ans =
     3
```

We can just as easily take the square root of each entry of a vector.

```
>> x=[0 1 4 9, 16, 25, 36]
x =
     0     1     4     9    16    25    36
>> y=sqrt(x)
y =
     0     1     2     3     4     5     6
```

We also saw that we can easily plot the results. The result of the following command is shown in **Figure 3.1**

```
>> plot(x,y,'*')
```

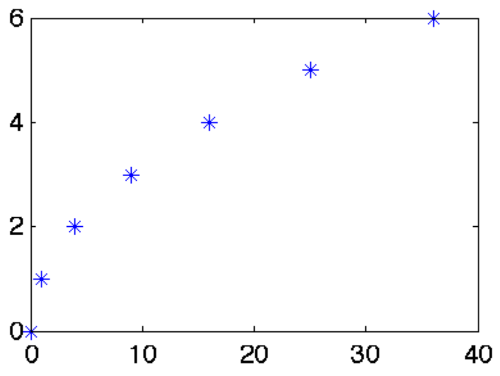


Figure 3.1. Plotting $y = \sqrt{x}$ at discrete values of x.

[1] Copyrighted material. See: http://msenux.redwoods.edu/Math4Textbook/

In this section, we will learn how to plot the graphs of a number of more complicated functions. We will also investigate a number of formatting options and we will spend some time learning how to annotate our plots (titles, labels, legends, etc.). Finally, in this section we gravitate away from the command line and use script files (introduced in the last section) to produce our plots.

Plotting Functions of a Single Variable

We begin by plotting a number of functions of a single variable in the Cartesian plane. Let's start by plotting the graph of a quadratic function.

▶ **Example 1.** *Plot the graph of* $y = x^2 - 2x - 3$.

When you were first introduced to drawing the graphs of functions in college algebra, your were probably taught the following standard technique. First create a table of points that satisfy the equation $y = x^2 - 2x - 3$, such as the one shown in **Table 3.1**(a). An arbitrary set of x-values are chosen from the function's domain and the function is evaluated at each value of x, as shown in the second column of **Table 3.1**(a). The results are recorded as ordered pairs in **Table 3.1**(b).

x	$y = x^2 - 2x - 3$
-2	$y = (-2)^2 - 2(-2) - 3$
-1	$y = (-1)^2 - 2(-1) - 3$
0	$y = (0)^2 - 2(0) - 3$
1	$y = (1)^2 - 2(1) - 3$
2	$y = (2)^2 - 2(2) - 3$
3	$y = (3)^2 - 2(3) - 3$
4	$y = (4)^2 - 2(4) - 3$

(a)

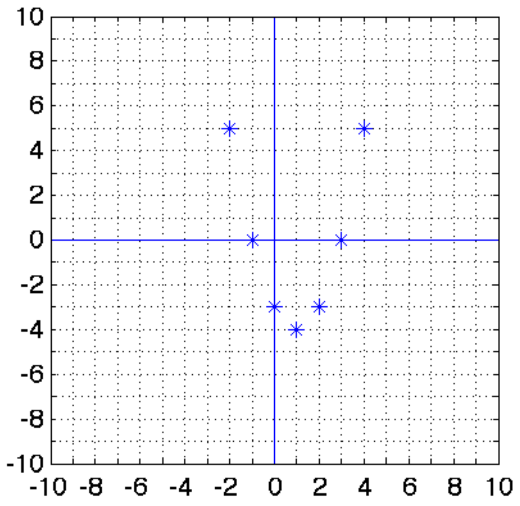
x	y	(x, y)
-2	5	$(-2, 5)$
-1	0	$(-1, 0)$
0	-3	$(0, -3)$
1	-4	$(1, -4)$
2	-3	$(2, -3)$
3	0	$(3, 0)$
4	5	$(4, 5)$

(b)

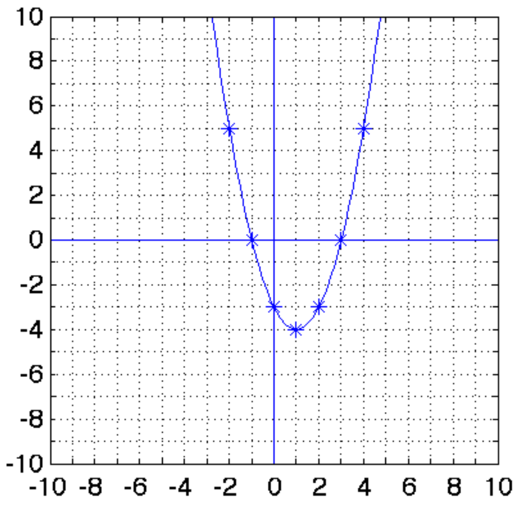
Table 3.1. Points satisfying the equation $y = x^2 - 2x - 3$.

Plotting the pairs in **Table 3.1**(b) provides a rough idea of the shape of the graph of $y = x^2 - 2x - 3$ in **Figure 3.2**(a). If we continue to plot all of the points that satisfy the equation $y = x^2 - 2x - 3$, we intuit that the final result will have the form shown in **Figure 3.2**(b).

To plot the graph of $y = x^2 - 2x - 3$ using Matlab, we follow roughly the same procedure. We load the x-values -2, -1, 0, 1, 2, 3, and 4 into a column vector **x**.



(a)



(b)

Figure 3.2. Plotting the graph of $y = x^2 - 2x - 3$.

```
>> x=(-2:4).'
x =
    -2
    -1
     0
     1
     2
     3
     4
```

To evaluate $y = x^2 - 2x - 3$ for each entry in the vector $\mathbf{x}$, we need to use array operations. We claim that the Matlab expression **y=x.^2-2*x-3** will evaluate the function $y = x^2 - 2x - 3$ at each entry of the column vector $\mathbf{x}$. To verify this claim, we present the following derivation.

$$\mathbf{y} = \mathbf{x}\,.\hat{}\,2 - 2 * x - 3 = \begin{bmatrix} -2 \\ -1 \\ \vdots \\ 4 \end{bmatrix} .\hat{}2 - 2 * \begin{bmatrix} -2 \\ -1 \\ \vdots \\ 4 \end{bmatrix} - 3$$

The array operation **.^2** will raise each element in the vector $\mathbf{x} = [-2, -1, \ldots, 4]^T$ to the second power. The product of the scalar 2 and the vector $\mathbf{x} = [-2, -1, \ldots, 4]^T$ in the second term is found by multiplying each element of the vector $\mathbf{x}$ by 2.

$$\mathbf{y} = \begin{bmatrix} (-2)^2 \\ (-1)^2 \\ \vdots \\ (4)^2 \end{bmatrix} - \begin{bmatrix} 2(-2) \\ 2(-1) \\ \vdots \\ 2(4) \end{bmatrix} - 3$$

Recall that Matlab subtracts 3 from a vector by subtracting 3 from each element of the vector. Thus, the vector **y** contains the entries

$$\mathbf{y} = \begin{bmatrix} (-2)^2 - 2(-2) - 3 \\ (-1)^2 - 2(-1) - 3 \\ \vdots \\ (4)^2 - 2(4) - 3 \end{bmatrix}.$$

Note that these are the same values of y generated in the second column of **Table 3.1**(a). Thus, it should come as no surprise that the following Matlab command generates the y-values in the second column of **Table 3.1**(b).

```
>> y=x.^2-2*x-3
y =
     5
     0
    -3
    -4
    -3
     0
     5
```

We can now use Matlab to plot the ordered pairs (x, y). The following command will generate the image shown in **Figure 3.3**(a).

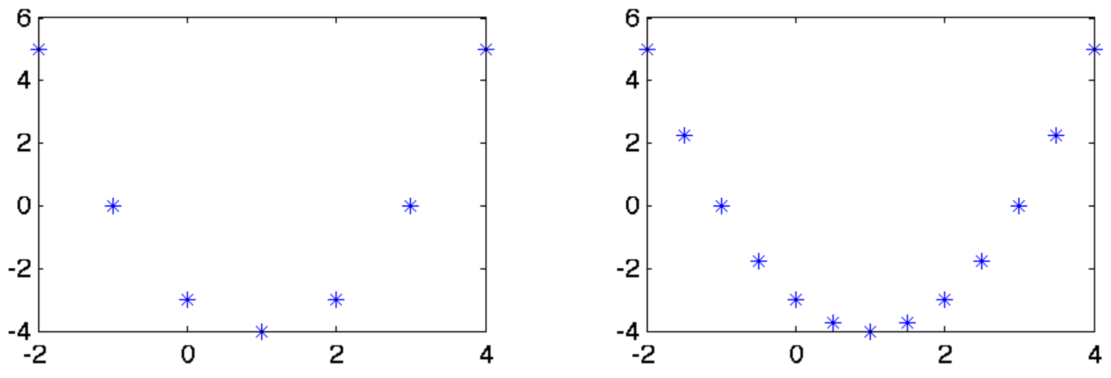
```
>> plot(x,y,'*')
```

We can increase the number of plotted points as follows. Note that if you change the vector **x**, you must recompute the vector **y**.

```
>> x=-2:0.5:4;
>> y=x.^2-2*x-3;
```

The following command will produce the image shown in **Figure 3.3**(b).

```
>> plot(x,y,'*')
```



(a) (b)

Figure 3.3. Using Matlab to plot the graph of $y = x^2 - 2x - 3$.

Formatting Options. Matlab's **plot** command offers a number of formatting options, some of which are listed in **Table 3.2**. For a full list of plotting options, type **help plot** and read the help file.

symbol	color	symbol	marker	symbol	linestyle
b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	–	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		

Table 3.2. Formatting options for Matlab's **plot** command

Matlab's **plot** command uses the syntax **plot(x,y,s)**, where *s* is a one, two, or three character string composed of the symbols in **Table 3.2**. For example, say you want red circles as markers and you want the markers connected with dotted lines. This is accomplished with the following command. The plot produced by the command is shown in **Figure 3.4**(a).

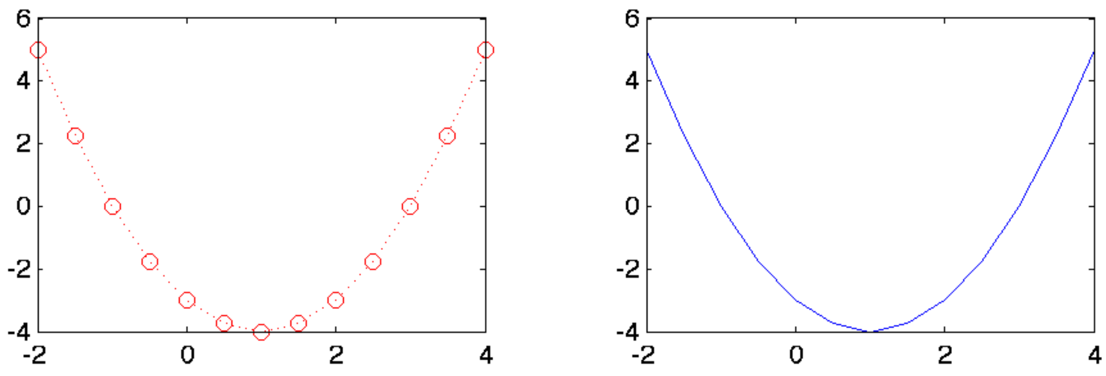
```
>> plot(x,y,'ro:')
```

You can plot the graph as an “almost smooth” curve if you create a vector **x** that contains a lot of points. Don’t forget to recalculate the vector **y**.

```
>> x=linspace(-2,4,200);
>> y=x.^2-2*x-3;
```

The following **plot** command can now be used to create the graph of $y = x^2 - 2x - 3$ in **Figure 3.4**(b).

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



(a) (b)

Figure 3.4. Using different plot styles with Matlab.

The command **plot(x,y,’b-’)** chooses the color blue, no marker, and connects consecutive points with solid line segments.[2] Technically, this is not a curve (it’s a sequence of line segements), but the graph of $y = x^2 - 2x - 3$ has the appearance of a smooth curve because we’ve plotted a lot of points. In contrast, the “curve” in **Figure 3.4**(a) has a “jagged” appearance, because too few points were used to approximate the graph of $y = x^2 - 2x - 3$.

[2] Actually, this is the default behavior of the Matlab’s **plot** command. If you execute the command **plot(x,y)**, you will get the color blue, no markers, and consecutive points will be connected with solid line segments.

Let's look at another example.

▶ **Example 2.** *Use Matlab to draw the graph of the function* $y = 3xe^{-0.25x}$ *on the interval* $[-5, 25]$.

We'll begin by creating a table to evaluate the function $y = 3xe^{-0.25x}$ at the specified x-values. Each of -5, 0, 5, 10, 15, 20, and 25 are substituted into the function to produce the result in the second column of **Table 3.3**(a). The results are then simplified to produce the pairs in **Table 3.3**(b).

x	$y = 3xe^{-0.25x}$
-5	$y = 3(-5)e^{-0.25(-5)}$
0	$y = 3(0)e^{-0.25(0)}$
5	$y = 3(5)e^{-0.25(5)}$
10	$y = 3(10)e^{-0.25(10)}$
15	$y = 3(15)e^{-0.25(15)}$
20	$y = 3(20)e^{-0.25(20)}$
25	$y = 3(25)e^{-0.25(25)}$

(a)

x	y	(x, y)
-5	-52.3551	$(-5, -52.3551)$
0	0.0000	$(-1, 0)$
5	4.2976	$(5, 4.2976)$
10	2.4625	$(10, 2.4625)$
15	1.0583	$(15, 1.0583)$
20	0.4043	$(20, 0.4043)$
25	0.1148	$(25, 0.1148)$

(b)

Table 3.3. Points satisfying the equation $y = 3xe^{-0.25x}$.

We claim that the Matlab assignment **y=3*x.*exp(-0.25*x)** will perform the substitutions shown in the second column of **Table 3.3**(a). A derivation will help make this claim a bit more clear. First, create a vector **x** with the values in the first column in **Table 3.3**(a).

```
>> x=(-5:5:25).'
x =
    -5
     0
     5
    10
    15
    20
    25
```

Substitute this vector for **x** in the expression **3*x.*exp(-0.25*x)**.

$$\mathbf{y} = 3 * \mathbf{x} \,.\!* \exp(-0.25 * \mathbf{x}) = 3 * \begin{bmatrix} -5 \\ 0 \\ \vdots \\ 25 \end{bmatrix} .\!* \exp\left(-0.25 * \begin{bmatrix} -5 \\ 0 \\ \vdots \\ 25 \end{bmatrix} \right)$$

Distribute the scalars.

$$\mathbf{y} = \begin{bmatrix} 3(-5) \\ 3(0) \\ \vdots \\ 3(25) \end{bmatrix} .\!* \exp\left(\begin{bmatrix} -0.25(-5) \\ -0.25(0) \\ \vdots \\ -0.25(25) \end{bmatrix} \right)$$

Matlab's **exp** function is array smart and will take the exponential of each element of the vector.

$$\mathbf{y} = \begin{bmatrix} 3(-5) \\ 3(0) \\ \vdots \\ 3(25) \end{bmatrix} .\!* \begin{bmatrix} \exp(-0.25(-5)) \\ \exp(-0.25(0)) \\ \vdots \\ \exp(-0.25(25)) \end{bmatrix}$$

The last step requires array multiplciation. Hence, the operator **.*** is used.

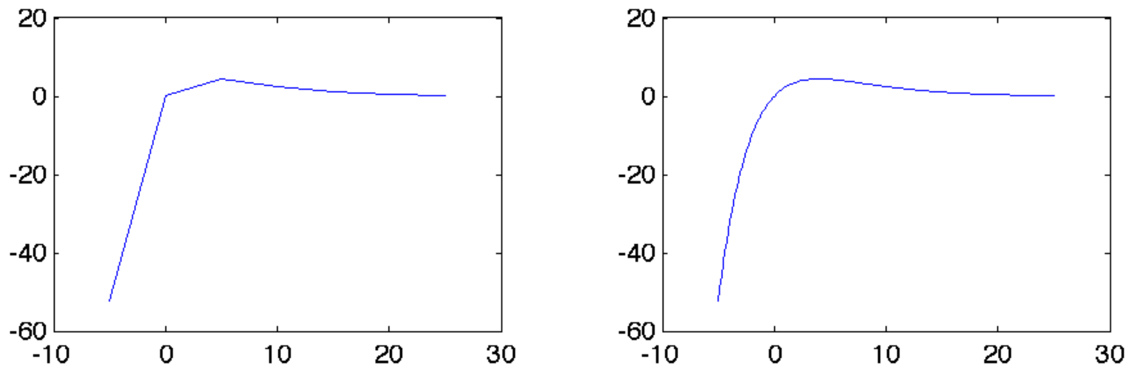
$$\mathbf{y} = \begin{bmatrix} 3(-5)\exp(-0.25(-5)) \\ 3(0)\exp(-0.25(0)) \\ \vdots \\ 3(25)\exp(-0.25(25)) \end{bmatrix}$$

Provided you take **3(-5)exp(-0.25(-5))** to mean $3(-5)e^{-0.25(-5)}$, each entry in this last vector is identical to the entries in column two of **Table 3.3**(a). Thus, it should come as no shock that the following command will produce a vector $\mathbf{y}$ identical to the second column of **Table 3.3**(b).

```
>> y=3*x.*exp(-0.25*x)
y =
  -52.3551
         0
    4.2976
    2.4625
    1.0583
    0.4043
    0.1448
```

It is now a simple matter to obtain a plot of $y = 3xe^{-0.25x}$. The following command will produce the plot shown in **Figure 3.5**(a).

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



(a) (b)

Figure 3.5. The graph of $y = 3xe^{-0.25x}$.

You'll note that the graph in **Figure 3.5**(a) has a severe case of the "Jaggies." That because we didn't plot enough points to emulate a smooth curve. However, this is easily rectified by adding move values to the vector **x** and recomputing the vector **y**. The following commands were used to produce the image in **Figure 3.5**(b).

```
>> x=linspace(-5,25,200);
>> y=3*x.*exp(-0.25*x);
>> plot(x,y)
```

A Note on Array Operators. In the Matlab expression **3*x.*exp(-0.25*x)**, note the intermingling of the scalar operator ***** and the array operator **.***. Here are some thoughts to keep in mind.

1. In the case of the expressions **3*x** and **-0.25*x**, we are multiplying a vector by scalars. This is a legal operation and is performed by multiplying each entry by a scalar.
2. Matlab functions are "array smart," so the Matlab expression **exp(-0.25*x)** causes Matlab to take the exponential of each element of the vector **-0.25x**.
3. Finally, each of the expressions **3*x** and **exp(-0.25*x)** are **vectors**! Therefore, it is not legal to take their product. Indeed, that is not what we want

to do anyway. What we want to do is to multiply the corresponding entries of each of these vectors. That is what array multiplication is for and that is why we use **.*** in the expression **3*x.*exp(-0.25*x)**.

Let's look at another example

▶ **Example 3.** *Draw the graph of* $y = 1/(1+99e^{-0.5t})$ *over the interval* $[0, 30]$.

Again, we evaluate the function at specified values of the requested domain. Substituting these values into the equation $y = 1/(1 + 99e^{-0.5t})$ produces the results shown in column 2 of **Table 3.4**(a). The results are then simplified to produce the pairs in **Table 3.4**(b).

t	$y = 1/(1+99e^{-0.5t})$
0	$y = 1/(1+99e^{-0.5(0)})$
10	$y = 1/(1+99e^{-0.5(10)})$
20	$y = 1/(1+99e^{-0.5(20)})$
30	$y = 1/(1+99e^{-0.5(30)})$

(a)

t	y	(t, y)
0	0.0100	$(0, 0.0100)$
10	0.5999	$(10, 0.5999)$
20	0.9955	$(20, 0.9955)$
30	1.0000	$(30, 1.0000)$

(b)

Table 3.4. Points satisfying the equation $y = 1/(1 + 99e^{-0.5t})$.

The Matlab expression **1./(1+99*exp(-0.5*t))** will produce the substitutions shown in the second column on **Table 3.4**(a). Again, a derivation will help make this clear. First, create a vector **t** with the values in the first column of **Table 3.4**(a).

```
>> t=(0:10:30).'
t =
     0
    10
    20
    30
```

Substitute this vector **t** in the expression **1./(1+99*exp(-0.5*t))**.

$$\mathbf{y} = 1 \,./ \left(1 + 99 * \exp\left(-0.5 * \begin{bmatrix} 0 \\ 10 \\ 20 \\ 30 \end{bmatrix}\right)\right)$$

Moving a little quicker with our explanation, first multiply the scalar -0.5 times each entry of the vector. Following that, **exp** is "array smart," taking the exponential of each element of the resulting vector.

$$\mathbf{y} = 1 \,./ \left(1 + 99 * \begin{bmatrix} \exp(-0.5(0)) \\ \exp(-0.5(10)) \\ \exp(-0.5(20)) \\ \exp(-0.5(30)) \end{bmatrix} \right)$$

Multiply each entry of the vector by 99. Recall that adding 1 to a vector causes Matlab to add 1 to each entry of that vector.

$$\mathbf{y} = 1 \,./ \begin{bmatrix} 1 + 99\exp(-0.5(0)) \\ 1 + 99\exp(-0.5(10)) \\ 1 + 99\exp(-0.5(20)) \\ 1 + 99\exp(-0.5(30)) \end{bmatrix}$$

Finally, the **./** array operator causes Matlab to divide the number 1 by each element of the array.

$$\mathbf{y} = \begin{bmatrix} 1/(1 + 99\exp(-0.5(0))) \\ 1/(1 + 99\exp(-0.5(10))) \\ 1/(1 + 99\exp(-0.5(20))) \\ 1/(1 + 99\exp(-0.5(30))) \end{bmatrix}$$

Provided you take **1/(1+99exp(-0.5(0)))** to mean $1/(1 + 99e^{-0.5(0)})$, each entry in this last vector is identical to the corresponding entry in column two of **Table 3.4**(a). Thus, the following command should produce a result that is identical to column two of **Table 3.4**(b).

```
>> y=1./(1+99*exp(-0.5*t))
y =
    0.0100
    0.5999
    0.9955
    1.0000
```

It is now a simple task to plot the graph of $y = 1/(1 + 99e^{-0.5t})$. The following command will provide the plot shown in **Figure 3.6**(a).

```
>> plot(t,y)
```

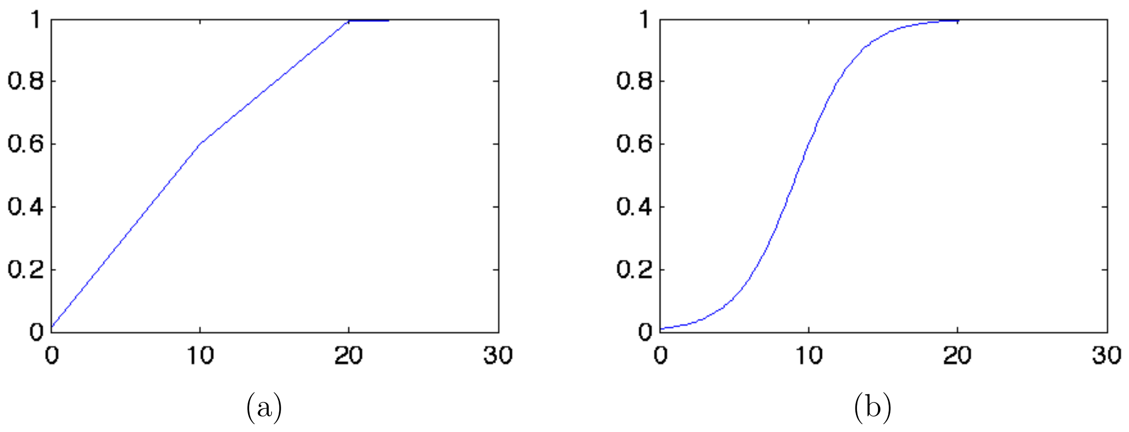


Figure 3.6. Sketch the graph of $y = 1/(1 + 99e^{-0.5t})$ on the interval $[0, 30]$.

Note that the plot in **Figure 3.6**(a) has a severe case of the "Jaggies" as we haven't plotted nearly enough points to give the graph a smooth appearance. This is easily fixed. Simply add more points to the vector **t**, recalculate **y**, then execute **plot(x,y)** to produce the image in **Figure 3.6**(b).

```
>> t=linspace(0,30,200);
>> y=1./(1+99*exp(-0.5*t));
>> plot(t,y)
```

Two or More Plots

It is not difficult to add a second plot to an existing figure window. One way to do this is with Matlab's **hold** command. Typing **hold** at the Matlab prompt is a toggle, which will turn holding on when it is off, and vice-versa. Hence, it is probably best to accentuate the desired result with either **hold on** or **hold off**.

When you type **hold on**, the plot is "held," and all subsequent calls to Matlab's **plot** command will add the new plot to the existing plot. Let's look at an example of this behavior in action.

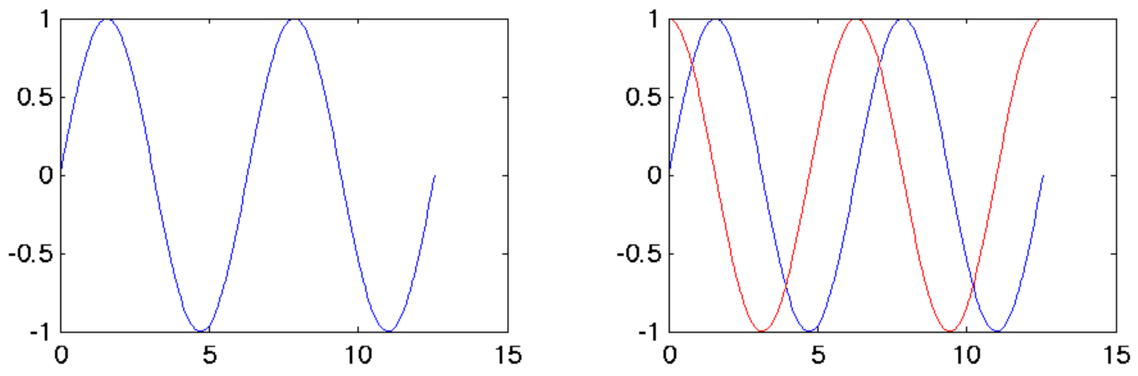
▶ **Example 4.** *Sketch the graphs of $y = \sin x$ and $y = \cos x$ over the interval* $[0, 4\pi]$.

We need to select enough x-values so that the graphs don't exhibit the "Jaggies."

```
>> x=linspace(0,4*pi,200);
```

Next, calculate the vector **y** and plot **y** versus **x** with the following commands. This will produce the image in **Figure 3.7**(a).

```
>> y=sin(x);
>> plot(x,y,'b-')
```



(a) (b)

Figure 3.7. Sketching the graphs of $y = \sin x$ and $y = \cos x$ over $[0, 4\pi]$.

Now, "hold" the graph with Matlab's **hold on** command.

```
>> hold on
```

Any subsequent plots will now take place in this "held" figure window.[3] Hence, if we calculate **y=cos(x)** and plot the results, the plot is superimposed on the "held" figure window. The result is shown in **Figure 3.7**(b)

```
>> y=cos(x);
>> plot(x,y,'r-')
```

When you look at the image in **Figure 3.7**(b), one difficulty becomes immediately apparent. That is, it is hard to tell which graph goes with which equation?

[3] Unless some other figure window is the currently active figure window. More on this later.

Matlab's **legend** command will come to the rescue in this situation. The following command was used to produce the legend shown in **Figure 3.8**.

```
>> legend('y = sin(x)','y = cos(x)')
```

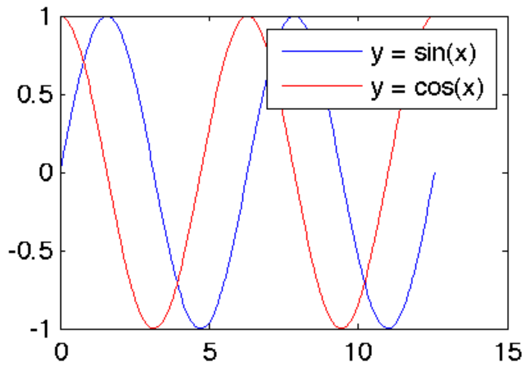


Figure 3.8. Adding a legend to help distinguish the plots.

Alternative Method --- Script File. In **Figure 3.8**, the addition of the legend allows the reader to differentiate between the two curves; the one in the color blue is the sine, the one in red is the cosine. Unfortunately, if you print this file in black and white, the use of color is not much help. Both curves will appear as solid black lines. In this alternate approach, we'll use different line styles to differentiate between the sine and cosine curves.

Moreover, we'll also avoid the use of Matlab's **hold on** command and demonstrate alternate methods for superimposing two or more plots on the same axes.

We'll find that script files are much more efficient when we have to execute a significant number of commands. As the number of commands required for a plot increases, you'll quickly tire of typing commands at the Matlab propmpt in the command window.

Again, the goal is to draw the graphs of two equations on the same axes, $y = \sin(x)$ and $y = \cos x$. Only this time we'll plot the functions on the domain $[-2\pi, 2\pi]$. Open the editor with this command.

```
>> edit
```

Enter the following lines in the editor.

```
% twoplot.m (Version 1.0  2/5/07)
%
% This script file plots the graphs of y = sin(x) and
% y = cos(x) on the same axes.

close all
x1 = linspace(-2*pi, 2*pi, 200);
y1 = sin(x1);
x2 = x1;
y2 = cos(x1);
plot(x1, y1, 'k-', x2, y2, 'k--')
axis tight
legend('y = sin(x)', 'y = cos(x)', 'Location', 'SouthWest')
```

Save the script file as **twoplot.m**. While in the editor, press F5, accept a change of the current directory if prompted, then hit OK to execute the file. The resulting plot is shown in **Figure 3.9**(a).

Some explanatory remarks are in order.

1. Note that the first four lines in the script begin with a %. This is a comment. Anything after the % is ignored by Matlab. You should get into the practice of sprinkling your code with comments.
2. The command **close all** will close **all** open figure windows, allowing a new figure window to pop up when the script executes.
3. Note how the x- and y-values for the sine were saved in x_1 and y_1, while the x- and y-values for the cosine are saved in x_2 and y_2.
4. The syntax for the **plot** command allows for more than one plot. Ideally, the syntax is **plot(x1,y1,s1,x2,y2,s2,x3,y3,s3,...)**, where the x_i's and y_i's contain the x- and y-data and each s_i contains a format string for the ith plot.
5. In **Figure 3.9**(a), note how the **axis tight** command in the script "tightens" the plot, at least when compared with the plot in **Figure 3.8**.
6. The legend command allows you to control where the legend is placed, in this case the southwest corner of the figure. Type **help legend** for more information.

Now, add the following three lines to the end of your script file and press F5 to save and execute in again. The resulting plot is shown in **Figure 3.9**(b). Note that **xlabel** and **label** take strings as input (delimited by single apostrophes) and use the strings to label the horizontal and vertical axes, respectively. The **title** command takes a string as input and uses it as a title for the plot. These annotations are shown in **Figure 3.9**(b).

```
xlabel('x-axis')
ylabel('y-axis')
title('Plotting the sine and cosine.')
```

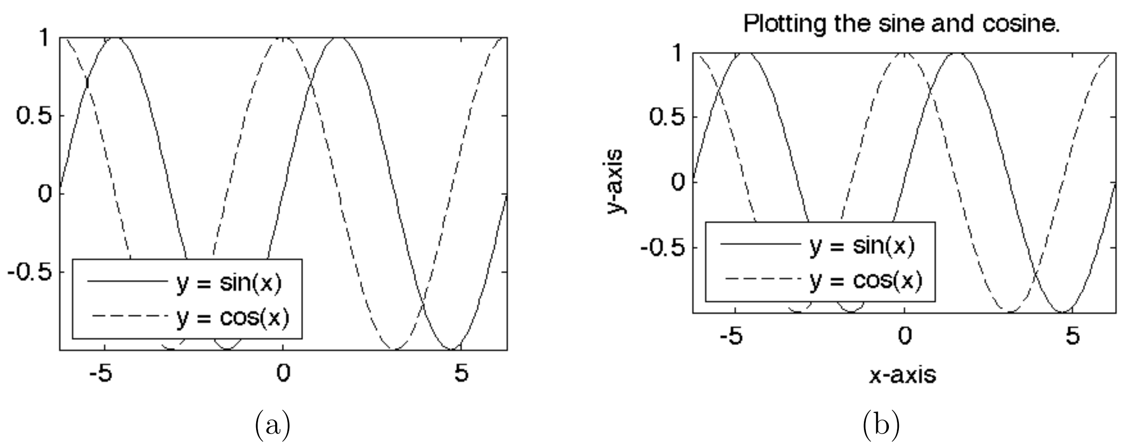


Figure 3.9. Superimposing two plots on one axes with a legend.

Finally, let's use Matlab's **line** command to add a pair of axes. We'll also add a grid with Matlab's **grid on** command. Add these final lines to your script file.

```
x=[-2*pi, 2*pi]; y=[0, 0]; line(x,y) % horizontal axis
x=[0, 0]; y=[-1, 1]; line(x,y) % vertical axis
grid on
```

Press F5 in the editor to save and execute this script to produce the image in **Figure 3.10**.

Here is the full script.

```
% twoplot.m (Version 1.0 2/5/07)
%
% This script file plots the graphs of y = sin(x) and
% y = cos(x) on the same axes.

close all
x1 = linspace(-2*pi, 2*pi, 200);
y1 = sin(x1);
x2 = x1;
y2 = cos(x1);
plot(x1, y1, 'k-', x2, y2, 'k--')
axis tight
legend('y = sin(x)', 'y = cos(x)', 'Location', 'SouthWest')
xlabel('x-axis')
ylabel('y-axis')
title('Plotting the sine and cosine.')
x=[-2*pi, 2*pi]; y=[0, 0]; line(x,y) % horizontal axis
x=[0, 0]; y=[-1, 1]; line(x,y) % vertical axis
grid on
```

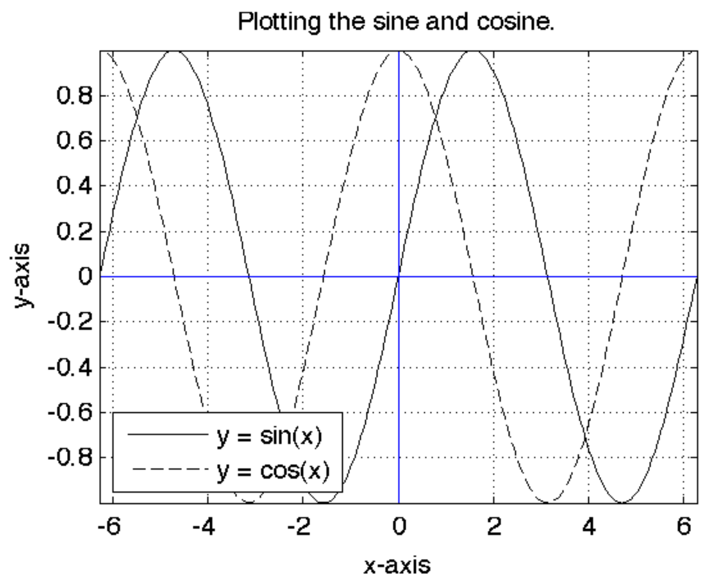


Figure 3.10. An annotated plot of sine and cosine.

3.1 Exercises

In **Exercises 1-6**, perform each of the following tasks. Type **help elfun** to find information on Matlab's elementary functions.

i. Set **x=linspace(a,b,n)** for the given values of a, b, and n.
ii. Calculate $y = f(x)$ for the given function f.
iii. Plot with **plot(x,y,s)** for the given formatting string **s**. In each exercise, explain the result of the formatting string.
iv. In each exercise, submit a printout of the plot and a printout of the script file that was used to draw the plot. Of course, if you are publishing to HTML, this is done for you.

1. $f(x) = \sin x$ $a = 0$, $b = 4\pi$, $n = 24$, **s = 'rs-'**

2. $f(x) = \cos^{-1} x$ $a = -1$, $b = 1$, $n = 24$, **s = 'mo'**

3. $f(x) = 2^x$ $a = -2$, $b = 4$, $n = 12$, **s = 'gd:'**

4. $f(x) = \sinh x$ $a = -5$, $b = 5$, $n = 20$, **s = 'kx--'**

5. $f(x) = \log_{10} x$ $a = 0.1$, $b = 10$, $n = 20$, **s = 'c--'**

6. $f(x) = \cosh^{-1} x$ $a = -5$, $b = 5$, $n = 20$, **s = 'b*-.'**

In **Exercises 7-12**, perform each of the following tasks.

i. Sketch the given function on a domain that shows all important features of the function (intercepts, extrema, etc.). Use enough points so that your plot takes the appearance of a "smooth curve."
ii. Label the horizontal and vertical axis with Matlab's **xlabel** and **ylabel** commands.
iii. Create a title with Matlab's **title** command that contains the equation of the function pictured in the plot.
iv. In each exercise, submit a printout of the plot and a printout of the script file that was used to draw the plot. Of course, if you are publishing to HTML, this is done for you.

7. $y = x^2 - 2x - 3$

8. $y = 5 - 2x - x^2$

9. $y = x^3 - 3x^2 - 28x + 60$

10. $y = -x^3 - 2x^2 + 29x + 30$

11. $y = x^4 - 146x^2 + 3025$

12. $y = 5184 - 180x^2 + x^4$

In **Exercises 13-18**, perform each of the following tasks.

i. Sketch the given function on the given domain. Use enough points so that your plot takes the appearance of a "smooth curve."
ii. Label the horizontal and vertical

axis with Matlab's **xlabel** and **ylabel** commands.

iii. Create a title with Matlab's **title** command that contains the equation of the function pictured in the plot.

iv. In each exercise, submit a printout of the plot and a printout of the script file that was used to draw the plot. Of course, if you are publishing to HTML, this is done for you.

13. $y = xe^{-x^2}$ on $[-5, 5]$

14. $y = \dfrac{1}{1+e^x}$ on $[-5, 5]$

15. $y = x \sin x$ on $[-12\pi, 12\pi]$

16. $y = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$ on $[-10, 10]$

17. $y = \dfrac{1}{1+x^2}$ on $[-10, 10]$

18. $y = (x^2 - 2x - 3)e^{-x^2}$ on $[-5, 5]$

In **Exercises 19-26**, perform each of the following tasks.

i. Sketch both of the given functions over the given domain on the same plot. Use different line styles for each plot. Provide a grid with **grid on**. Use enough points so that each of your plots has a "smooth" appearance.

ii. Label the horizontal and vertical axes appropriately and provide a title. Create a legend.

iii. In each exercise, submit a printout of the plot and a printout of the script file that was used to draw the plot. Of course, if you are publishing to HTML, this is done for you.

19. $y = \sin 2x$ and $y = \cos 2x$ on $[0, 4\pi]$.

20. $y = 3 \sin \pi x$ and $y = -2 \cos \pi x$ on $[-4, 4]$.

21. $\sin 2x$ and $y = 1/2$ on $[0, 2\pi]$.

22. $y = \cos(x/2)$ and $y = -1/2$ on $[0, 4\pi]$.

23. $y = \dfrac{2+3x^2}{1+x^2}$ and $y = 3$ on $[-10, 10]$.

24. $y = 3 - e^{-0.5x^2}$ and $y = 3$ on $[-3, 3]$.

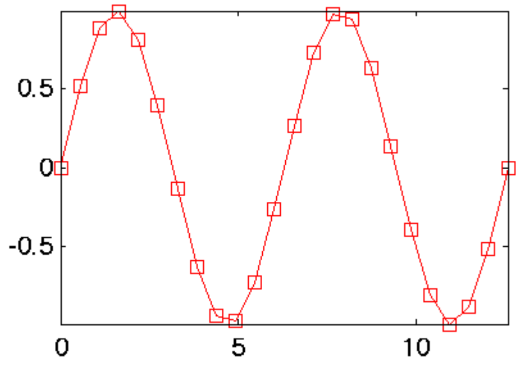
25. $y = (x-2)(3-x)(x+5)$ and $y = -2(x-2)(3-x)(x+5)$ on $[-10, 10]$

26. $y = (x+5)(3-x)$ and $y = -3(x+5)(3-x)$ on $[-10, 10]$

3.1 Answers

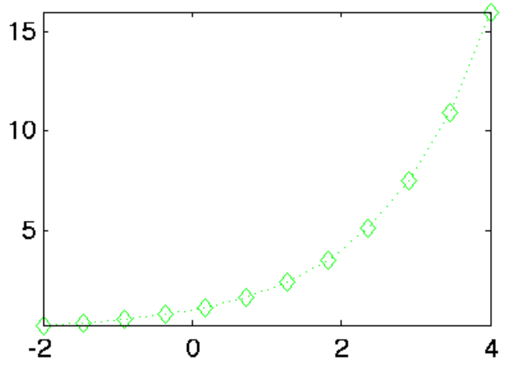
1. The following script file was used to produce the plot that follows.

```
x=linspace(0,4*pi,24);
y=sin(x);
plot(x,y,'rs-')
axis tight
```



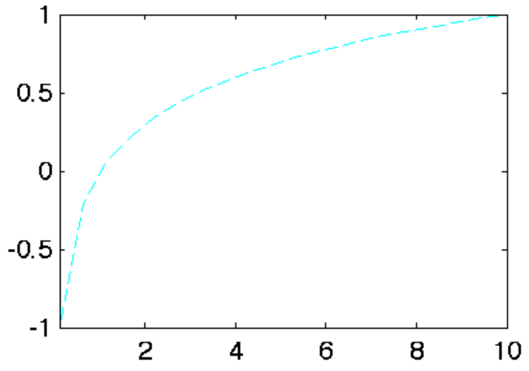
3. The following script file was used to produce the plot that follows.

```
x=linspace(-2,4,12);
y=2.^x;
plot(x,y,'gd:')
axis tight
```



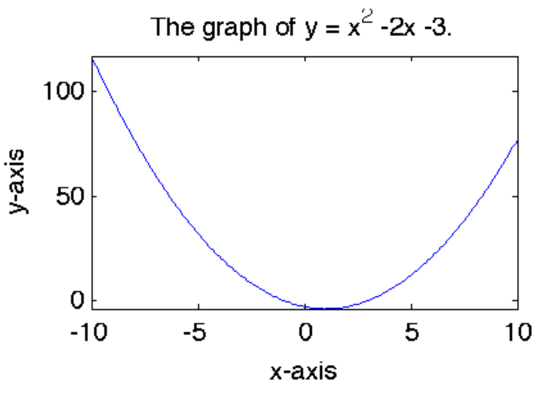
5. The following script file was used to produce the plot that follows.

```
x=linspace(0.1,10,20);
y=log10(x);
plot(x,y,'c--')
axis tight
```



7. The following script file was used to produce the plot that follows.

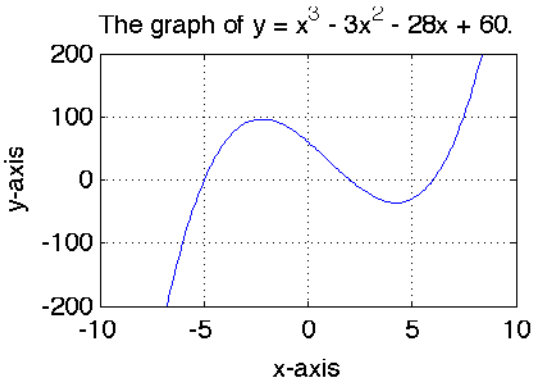
```
x=linspace(-10,10,200);
y=x.^2-2*x-3;
plot(x,y)
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('The graph of y = x^2
  - 2x - 3.')
```



9. The following script file was used to produce the plot that follows.

```
x=linspace(-10,10,200);
y=x.^3-3*x.^2-28*x+60;
plot(x,y)
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('The graph of y = x^3
  - 3x^2 - 28x + 60.')
axis([-10,10,-200,200])
grid on
```

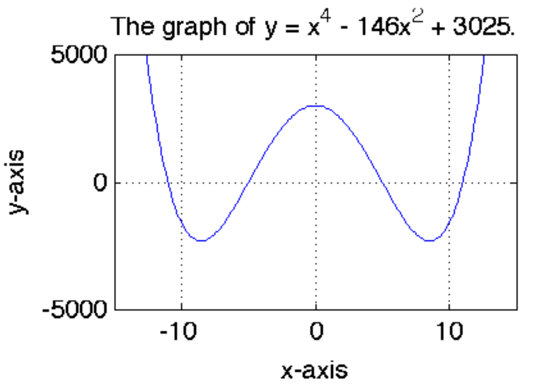
Note the use of Matlab's **axis** command and the fact that we've turned on the grid.



11. The following script file was used to produce the plot that follows.

```
x=linspace(-15,15,200);
y=x.^4-146*x.^2+3025;
plot(x,y)
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('The graph of
  y = x^4 - 146x^2 + 3025.')
axis([-15,15,-5000,5000])
grid on
```

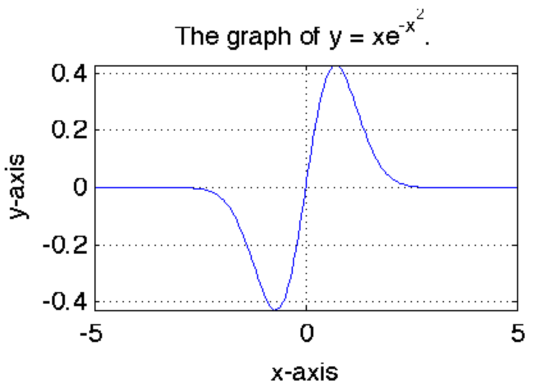
Note the use of Matlab's **axis** command and the fact that we've turned on the grid.



13. The following script file was used to produce the plot that follows.

```
x=linspace(-5,5,200);
y=x.*exp(-x.^2);
plot(x,y)
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('The graph of
  y = xe^{-x^2}.')
grid on
```

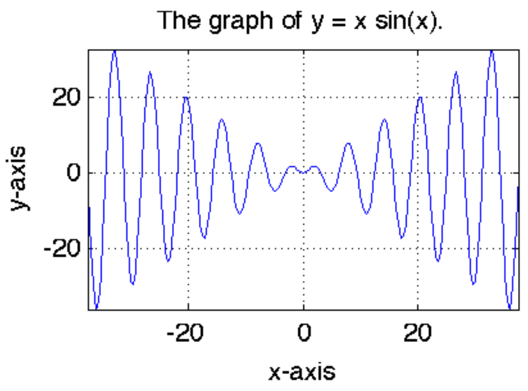
Note that we've turned on the grid.



15. The following script file was used to produce the plot that follows.

```
x=linspace(-12*pi,12*pi,200);
y=x.*sin(x);
plot(x,y)
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('The graph of
  y = x sin(x).')
grid on
```

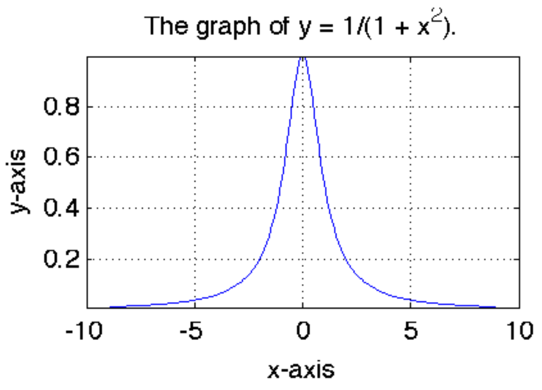
Note that we've turned on the grid.



17. The following script file was used to produce the plot that follows.

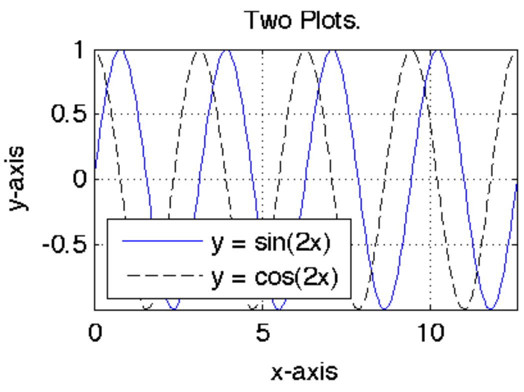
```
x=linspace(-10,10,200);
y=1./(1+x.^2);
plot(x,y)
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('The graph of
  y = 1/(1 + x^2).')
grid on
```

Note that we've turned on the grid.



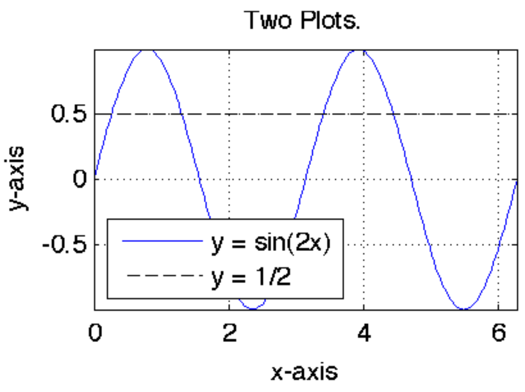
19. The following script file was used to produce the plot that follows.

```
x=linspace(0,4*pi,200);
y1=sin(2*x);
y2=cos(2*x);
plot(x,y1,'b-',x,y2,'k--')
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('Two Plots.')
grid on
legend('y = sin(2x)',
  'y = cos(2x)',
  'Location',
  'SouthWest')
```



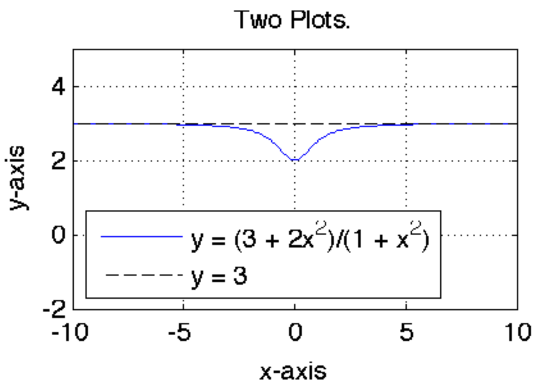
21. The following script file was used to produce the plot that follows.

```
x=linspace(0,2*pi,200);
y1=sin(2*x);
y2=1/2*ones(size(x));
plot(x,y1,'b-',x,y2,'k--')
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('Two Plots.')
grid on
legend('y = sin(2x)',
  'y = 1/2',
  'Location',
  'SouthWest')
```



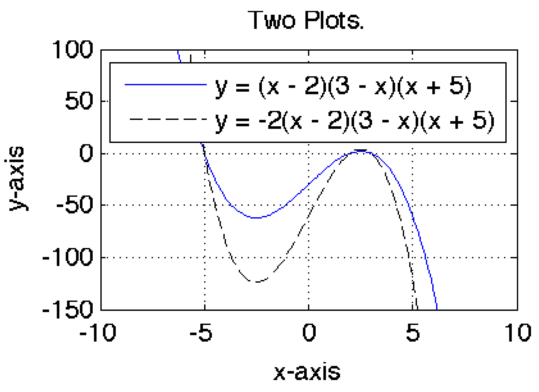
23. The following script file was used to produce the plot that follows.

```
x=linspace(-10,10,200);
y1=(2+3*x.^2)./(1+x.^2);;
y2=3*ones(size(x));
plot(x,y1,'b-',x,y2,'k--')
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('Two Plots.')
grid on
legend('y = (3 + 2x^2)
  /(1 + x^2)',
  'y = 3',
  'Location',
  'SouthWest')
axis([-10,10,-2,5])
```



25. The following script file was used to produce the plot that follows.

```
x=linspace(-10,10,200);
y1=(x-2).*(3-x).*(x+5);
y2=2*(x-2).*(3-x).*(x+5);;
plot(x,y1,'b-',x,y2,'k--')
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('Two Plots.')
grid on
legend('y = (x - 2)(3 - x)
  (x + 5)', 'y = -2(x - 2)
  (3 - x)(x + 5)')
axis([-10,10,-150,100])
```



3.2 Parametric and Polar Equations

In this section we will continue to pursue or exploration of curves drawn in the plane. We will look at three distinct categories of plane curves.

1. Parametric equations.
2. Polar equations.
3. Algebraic curves.

We begin with the first category.

Parametric Equations

As a first example, let's examine the curve defined by the following set of *parametric equations*.

$$\begin{aligned} x &= |1-t| \\ y &= |t|+2 \end{aligned} \tag{3.1}$$

Note that x and y are defined in terms of an independent parameter t.

The approach here is similar to the approach used to draw the graph of a function of one variable. First, create a table of points that satisfy the parametric equations. The usual technique is to select arbitrary values of the independent parameter t, then use the parametric **equations 3.1** to calculate x and y at each value of t, as shown in **Table 3.5**(a). The computations in **Table 3.5**(a) are simplified and summarized in **Table 3.5**(b).

t	$x = \|1-t\|$	$y = \|t\|+2$
-3	$x = \|1-(-3)\|$	$y = \|(-3)\|+2$
-2	$x = \|1-(-2)\|$	$y = \|(-2)\|+2$
-1	$x = \|1-(-1)\|$	$y = \|(-1)\|+2$
0	$x = \|1-0\|$	$y = \|0\|+2$
1	$x = \|1-1\|$	$y = \|1\|+2$
2	$x = \|1-2\|$	$y = \|2\|+2$
3	$x = \|1-3\|$	$y = \|3\|+2$

(a)

t	x	y
-3	4	5
-2	3	4
-1	2	3
0	1	2
1	0	3
2	1	4
3	2	5

(b)

Table 3.5. Creating tables of points that satisfy the parametric **equations 3.1**.

[4] Copyrighted material. See: http://msenux.redwoods.edu/Math4Textbook/

Matlab greatly eases the calculations shown in **Tables 3.5**(a) and (b). Create a column vector t ranging from -3 to 3 to match the values of t in **Table 3.5**(a). Compute vectors **x** and **y**.

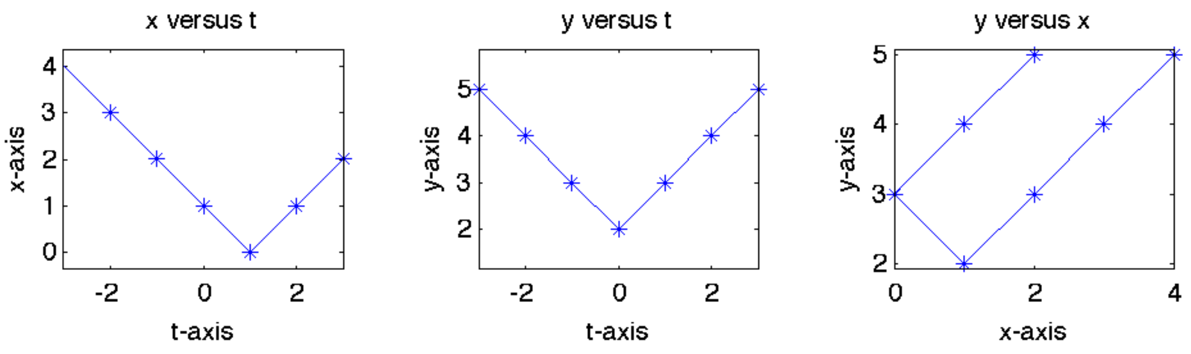
```
>> t=(-3:3).';
>> x=abs(1-t);
>> y=abs(t)+2;
```

We can build a matrix with the column vectors **t**, **x**, and **y** and display the results to the screen.

```
>> [t,x,y]
ans =
    -3      4      5
    -2      3      4
    -1      2      3
     0      1      2
     1      0      3
     2      1      4
     3      2      5
```

Note that these results match those in **Table 3.5**(b).

At this point, we have several choices: (1) we can plot **x** versus **t** (**Figure 3.11**(a)), or (2) we ccould plot **y** versus **t** (**Figure 3.11**(b)), or (3) we could plot **y** versus **x** (**Figure 3.11**(c)). Each figure has its advantages and disadvantages.



(a) **plot(t,x,'b*-')** (b) **plot(t,y,'b*-')** (c) **plot(x,y,'b*-')**

Figure 3.11. Plotting the data from **Table 3.5**(b).

The path in **Figure 3.11**(c) holds the most interest. If we think of (x, y) as the position of a particle in the plane at time t, then the path in **Figure 3.11**(c) describes the path taken by the particle in the plane over time.

Animating the Path of the Particle. Unfortunately, the path shown in **Figure 3.11**(c) is static, but we can rectify this with Matlab's **comet** command, which is used to animate the motion of the particle in the plane. Enter the following code into the Matlab editor. Execute the code, either with cell mode enabled in the editor, or as a script file.

```
close all
t=linspace(-3,3,1000);
x=abs(1-t);
y=abs(t);
comet(x,y)
```

If the animation takes inordinately long, you can stop the animation with `Ctrl+C` in the command window. Then try deleting the number of points in the **linspace** command (e.g., **x=linspace(-3,3,500)**). On the other hand, if the animation is too quick, increase the number of points in the **linspace** command (e.g., **linspace(-3,3,1500)**), then run the script again. The animation provided by Matlab's plot command is extremely valuable as it provides the user with a sense of the particle's motion in the plane over time.

Although we cannot display the animation provided by the **comet** in this document, we can indicate that the direction of motion is that shown in **Figure 3.12**.

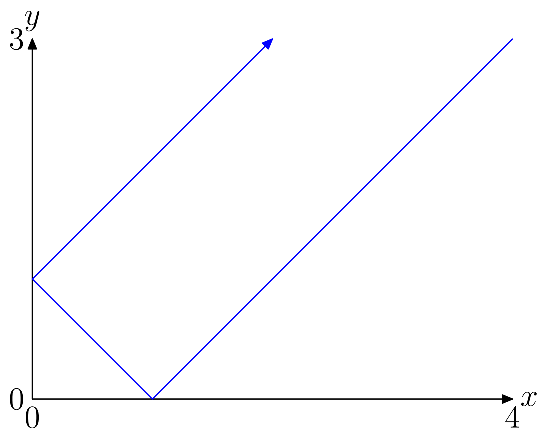


Figure 3.12. The **comet** command provides a sense of the direction of motion.

Let's look at another example.

▶ **Example 1.** *Consider the following set of parametric equations.*

$$\begin{aligned} x &= 3\cos 2t \\ y &= 5\sin 3t \end{aligned} \tag{3.2}$$

Plot x versus t, y versus t, then y versus x.

First, use the **linspace** command to create a vector of t-values. Then compute the vectors **x** and **y**. *Note: We assume that you are working either in cell-enabled mode in your editor, or you are placing the following commands in a script file.*

```
t=linspace(0,2*pi,500);
x=3*cos(2*t);
y=5*sin(3*t);
```

The following commands will produce the plot of x versus t in **Figure 3.13**(a).

```
plot(t,x)
axis tight
xlabel('t-axis')
ylabel('x-axis')
title('Plotting x versus t.')
```

The next set of command will produce the plot of y versus t in **Figure 3.13**(b).

```
plot(t,y)
axis tight
xlabel('t-axis')
ylabel('y-axis')
title('Plotting y versus t.')
```

Finally, the following commands will produce a plot of y versus x in **Figure 3.13**(c).

```
plot(x,y)
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('Plotting y versus x.')
```

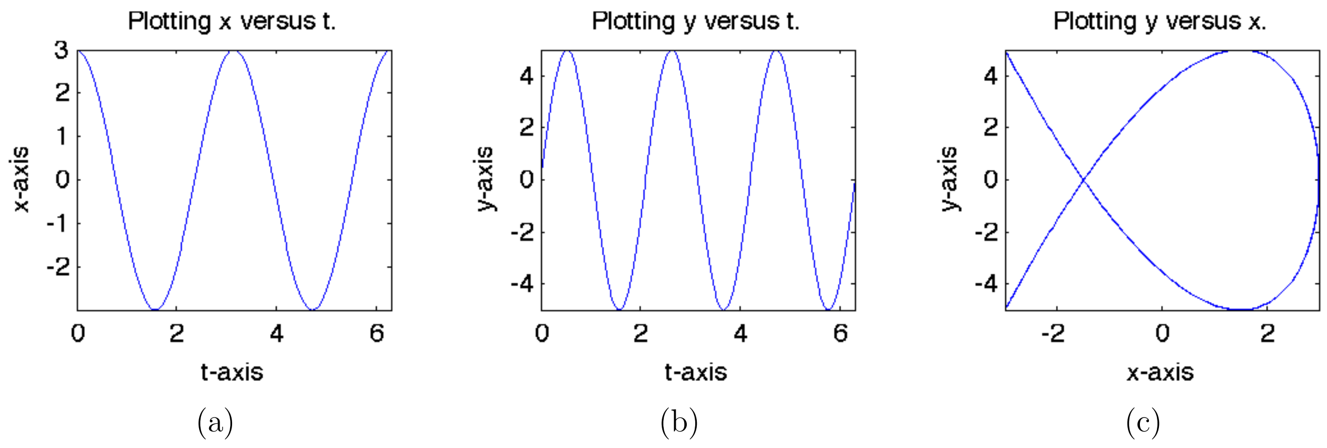


Figure 3.13. Plotting x and y versus t.

However, the static nature of each of the graphs in **Figure 3.13** don't reveal the character of the motion of the particle in the plane, as dictated by the parametric equations **3.2**. Only the **comet** command can reveal the motion. Although we cannot portray the dynamic motion of the particle in this printed document, you should try the following command to get a sense of the motion of the particle in the plane.

```
comet(x,y)
```

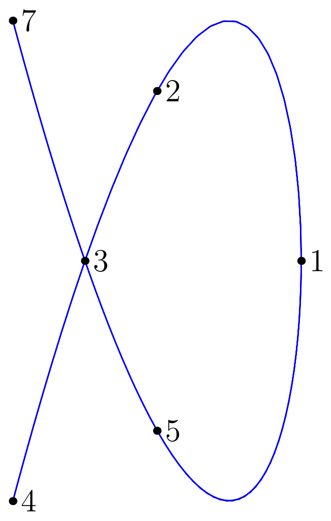


Figure 3.14. Indicating the motion of the particle using the **comet** command.

The **comet** command will show that the particle starts at postion 1 in **Figure 3.14**. It then begins its motion counterclockwise, traveling through the points at 2, 3 and stopping momentarily at the point at 4. It then retraces its route, returning through 3 and 2 to its original point at 1. It continues its clockwise journey through 5, passes through the point 3 again and stops momentarily at the point at 7. It then retraces its route back through 3 and 5, returning to its starting point at 1.

Polar Equations

The Cartesian coordinate system is not the only way to determine the position of a point in the plane. Rather than giving its x- and y-coordinates, we can determine the location of a point in the plane by indicating an angle and radial length, called *polar coordinates*, as shown in **Figure 3.15**.

We must make two important points regarding polar coordinates.

1. Angles are measured from the positive x-axis, often called the *polar axis*. The counterclockwise direction marks positive angles, the clockwise direction marks negative angles.
2. The value of r can be positive or negative. A negative r-value indicates a reversal of direction when measuring the radial distance. You "go backwards," so to speak.

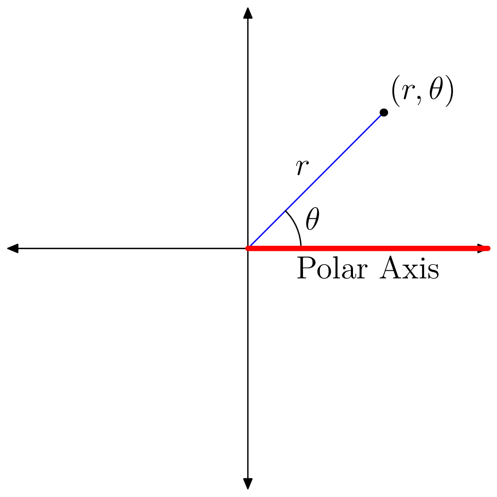


Figure 3.15. Defining polar coordinates.

Plotting points in polar coordinates is best accomplished on polar coordinate graph paper, an example of which is shown in **Figure 3.16**. To plot the point

$Q(1, 60°)$, you rotate $60°$ in the counterclockwise direction from the polar axis, then move 1 unit outward in the radial direction to obtain the point $Q(1, 60°)$ shown in **Figure 3.16**. In similar fashion, rotate $-60°$ in a clockwise direction from the polar axis, then move 1 unit outward in the radial direction to obtain the point $R(1, -60°)$ shown in **Figure 3.16**. Finally, to understand how to proceed when r is negative, consider the point $S(-1, 30°)$. To plot this point, first rotate $30°$ in a counterclockwise direction from the polar axis, but then reverse the radial direction, moving "backward 1 unit" to obtain the point $S(-1, 30°)$ shown in **Figure 3.16**.

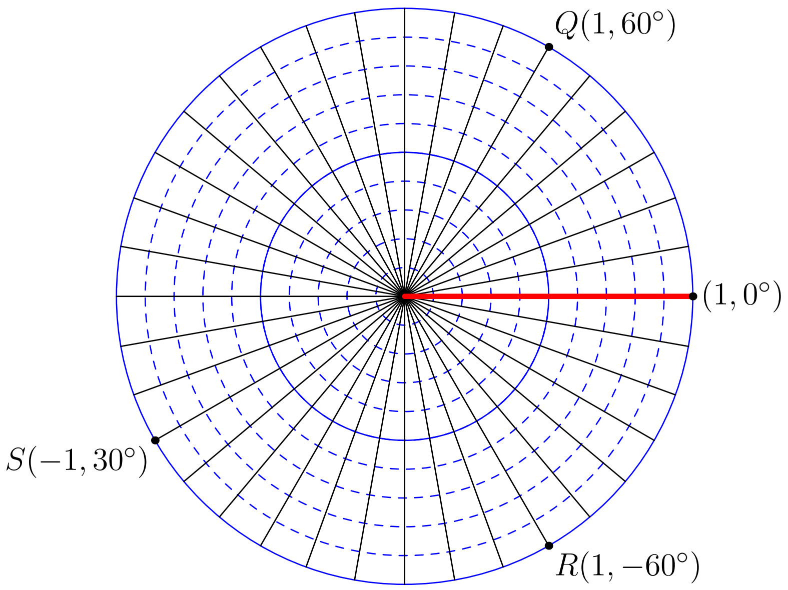


Figure 3.16. Polar coordinate graph paper.

The difficulty with polar coordinates is that unlike Cartesian coordinates, points can have multiple representations. For example, the point Q could also be the point $Q(1, 420°)$, the point $Q(-1, 240°)$, or even the point $Q(1, -300°)$. Multiple representations can cause difficulty in trigonometry and calculus, but we won't worry much about it in this course.

Matlab fully understands polar coordinates. It has commands **cart2pol** and **pol2cart** to transform Cartesian coordinates to polar coordinates, and vice-versa. As we are most intersted in the plotting of polar equations, we begin by first investigating Matlab's **polar** command.

▶ **Example 2.** *Sketch the graph of the polar equation $r = 2\sin 2\theta$.*

Note that the radial length r is given as a function of the angle θ. This is typical of the form of most polar equations. To use Matlab's **polar** command to plot the graph of $r = 2\sin 2\theta$, use the following commands, either in a script or in the cell-enabled editor. Executing these commands produces the *four-leave rose* shown in **Figure 3.17**.

```
theta=linspace(0,2*pi,200);
r=2*sin(2*theta);
polar(theta,r)
```

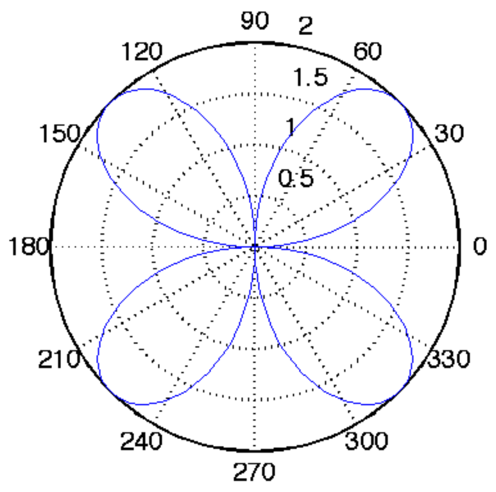


Figure 3.17. A four-leaf rose.

Note that Matlab's **polar** command plots the rose on a form of polar coordinate graph paper.

Polar to Cartesian Coordinates. In order to obtain more control over plots of polar equations, we will need to know how to transform polar coordinates into Cartesian coordinates. To do so, we need to understand a basic trigonometric definition. Consider the image in **Figure 3.18**, where the polar angle and radius are θ and r, respectively. In addition, we've labeled the x- and y-axes and the point $P(x, y)$.

Here are the standard trigonometric definitions of sine and cosine.

Definition 3. *Given the radius r and polar angle θ, we determine the Cartesian coordinates of the point P as follows.*

$$x = r\cos\theta \qquad \text{and} \qquad y = r\sin\theta \tag{3.3}$$

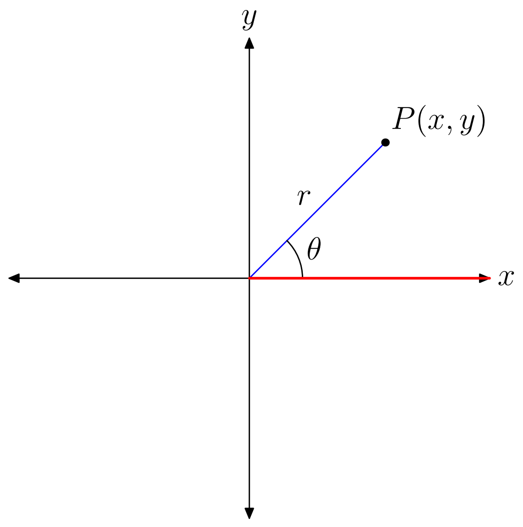


Figure 3.18. Transforming polar to Cartesian coordinates.

We can use **Definition 3** to change the polar form of the rose in **Example 2** to Cartesian form.

▶ **Example 4.** *Use Matlab to sketch the graph of the polar equation $r = 2\sin 2\theta$ by first transforming polar coordinates to Cartesian form.*

We can compute vectors **r** and θ as before.

```
theta=linspace(0,2*pi,200);
r=2*sin(2*theta);
```

Next, note that **Definition 3** provides

$$x = r\cos\theta \qquad \text{and} \qquad y = r\sin\theta.$$

We use this definition to change to Cartesian coordinates.

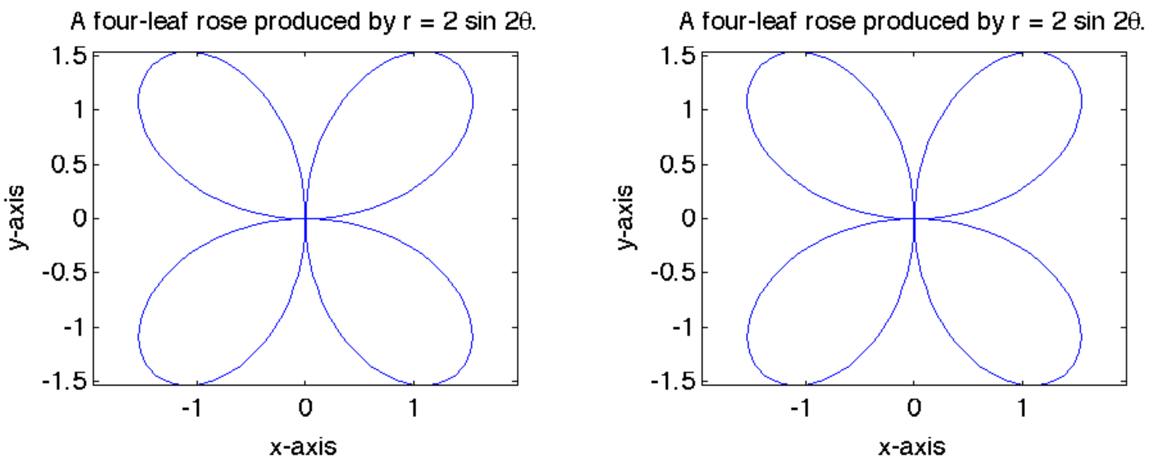
```
x=r.*cos(theta);
y=r.*sin(theta);
```

The **plot** command will now produce the image shown in **Figure 3.19**(a). We've added a command **axis equal** which sets the aspect ratio so that equal tick mark increments on the x- and $y-$ axes are equal in size.

```
plot(x,y)
axis equal
xlabel('x-axis')
ylabel('y-axis')
title('A four-leaf rose produced by r = 2 sin 2\theta.')
```

Matlab's **pol2cart** will handle the change to Cartesian coordinates if you wish to use that instead. The following commands produce an identical rose in **Figure 3.19**(b).

```
theta=linspace(0,2*pi,200);
r=2*sin(2*theta);
[x,y]=pol2cart(theta,r);
plot(x,y)
axis equal
xlabel('x-axis')
ylabel('y-axis')
title('A four-leaf rose produced by r = 2 sin 2\theta.')
```



(a) $x = r\cos\theta$ and $y = r\sin\theta$. (b) **pol2cart**

Figure 3.19. The four-leaf rose in Cartesian coordinates.

The Dynamics are Missing. The static result in **Figures 3.19**(a) and (b) hide the motion of the particle in the plane. We strongly encourage our readers to try the following code to get a feel for the motion of the particle as it traces the rose.

```
theta=linspace(0,2*pi,200);
r=2*sin(2*theta);
[x,y]=pol2cart(theta,r);
comet(x,y)
```

Algebraic Curves

Often, we are presented with an equation in x and y where it would be difficult or impossible to solve for one variable in terms of the other. In this case, the techniques for plotting that we've provided thus far will not work. However, the class of *algebraic curves* is a fascinating subject, so we would like to develop technique for plotting members of this class.

Matlab has a suite of functions called the *EZ* functions (short for "Easy") that will provide a quick sketch with a minimum of preparation. They don't offer the fine-grained control of the **plot** command, but if your goal is a simple quick sketch, they will suffice for that purpose. Here are the members of the "ez-suite."

ezcontour	**ezmeshc**	**ezpolar**
ezcontourf	**ezplot**	**ezsurf**
ezmesh	**ezplot3**	**ezsurfc**

We will concentrate on using the **ezplot** command to sketch the graphs of algebraic curves, but we encourage our readers to try some of the other commands. Help is available for each command in the form of **help ezplot**. For example, if you want help for drawing contours or level curves, you will find the information you need with the command **help ezcontour**.

Let's look at an example of an algebraic curve.

▶ **Example 5.** *Sketch the graph of the* lemniscate *given by the equation*

$$(x^2 + y^2)^2 = 2(x^2 - y^2). \tag{3.4}$$

First, note that solving for one variable in terms of the other is a seemingly impossible task (or at least extremely difficult). We will use the implicit nature of the **ezplot** command to sketch a graph of equation (3.4). As the **ezplot** command is meant as a quick hack, we'll return to entering commands at the prompt in the command window.

First, set one side of the equation equal to zero, as **ezplot**'s implicit plotting routine expects a function of the form $f(x, y) = 0$.

$$(x^2 + y^2)^2 - 2(x^2 - y^2) = 0 \tag{3.5}$$

The **ezplot** routine allows us to enter the left-hand side of **equation (3.5)** as a string (remember that strings are delimited by single apostrophes). The following command will produce the image in **Figure 3.20**(a).

```
>> ezplot('(x^2+y^2)^2-2*(x^2-y^2)')
```

Three facts warrant our attention.

1. When the implicit function is entered as a string, as it is above, Matlab vectorizes the string (makes it "array smart") before passing it along to whatever routine is used to produce the curve. That is, Matlab sends along the string **'(x.^2+y.^2).^2-2*(x.^2-y.^2)'**
2. With the usage above, the **ezplot** routine uses a default domain $[-2\pi, 2\pi]$ and range $[-2\pi, 2\pi]$.
3. The **ezplot** routine labels axes and provides a title automatically. It determines the variables and title by parsing the string passed to the routine.

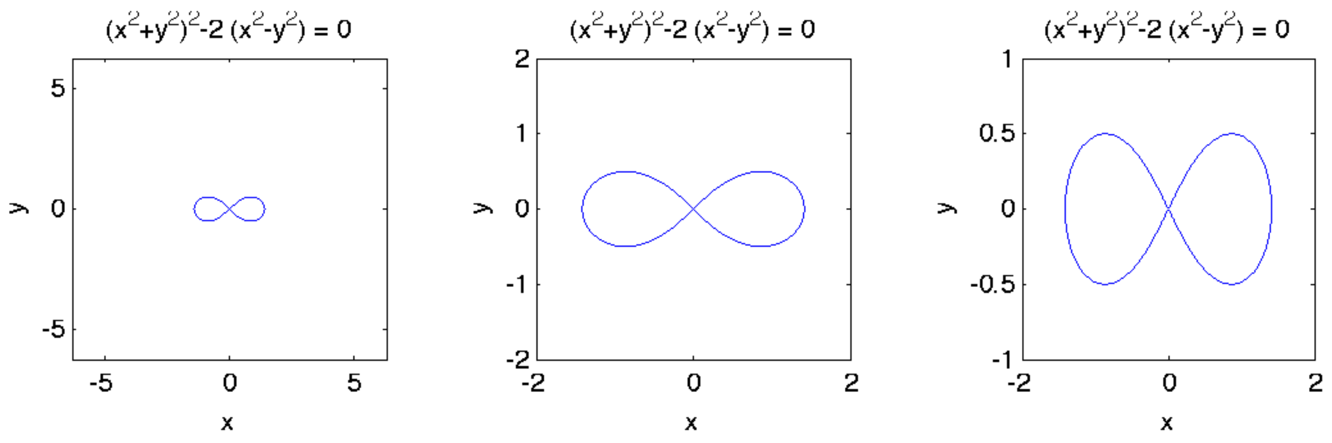
We can change the domain and range if we wish. The following command will produce the image in **Figure 3.20**(b).

```
>> ezplot('(x^2+y^2)^2-2*(x^2-y^2)', [-2,2])
```

In this form, $[-2, 2]$ is used for both the domain and range. The effect is seen by examining the scale on the axes in **Figure 3.20**(a). Note that both domain and range are set to $[-2, 2]$.

The domain and range can be set independently. The following command will produce the image in **Figure 3.20**(c). Note that the domain is $[-2, 2]$, but the range is now set to $[-1, 1]$.

```
>> ezplot('(x^2+y^2)^2-2*(x^2-y^2)',[-2,2,-1,1])
```



(a) (b) (c)

Figure 3.20. Changing domain and range.

Anonymous Functions. Matlab's **ezplot** routine will accept an anonymous function in lieu of a string. You can create an anonymous function for the lemniscate with the following command.

```
>> f = @(x,y) (x.^2+y.^2).^2-2*(x.^2-y.^2);
```

Anonymous functions are powerful constructs that were first introduced in Matlab 7. We'll have much more to say about them as the course progresses, but let's first comment on the construction of this particular anonymous function.

1. The variable **f** is assigned a *function handle*, which is essentially a number that identifies where to look in the memory of your machine for the function definition.
2. Next comes the function handle symbol **@**.
3. The **@** symbol is followed (in parentheses) by the input variables for the anonymous function. The order is important here, at least as far as the **ezplot** command is concerned. The first input variable is placed on the horizontal axis, the second on the vertical axis.
4. The input variables are followed by the function definition; i.e., the output value of the function.
5. As we saw above, **ezplot** will vectorize strings and make them "array smart." This is not the case with anonymous functions used with **ezplot**, so one must use array operators to make the anonymous function definition "array smart."

Whenever you create an anonymous function at the command prompt, it is a good idea to test the function. Note that if we substitute $x = 1$ and $y = 2$, the result should be

$$(x^2+y^2)^2-2(x^2-y^2)=(1^2+2^2)^2-2(1^2-2^2)=25+6=31.$$

If we evaluate the anonymous function **f** at the point $(1,2)$, we see that it is working properly.

```
>> f(1,2)
ans =
    31
```

Note that Matlab uses the function notation of mathematics to evaluate the anonymous function at the given values of x and y.

All looks well, so let's create the graph in **Figure 3.21** with the following command.

```
>> ezplot(f,[-2,2,-1,1])
```

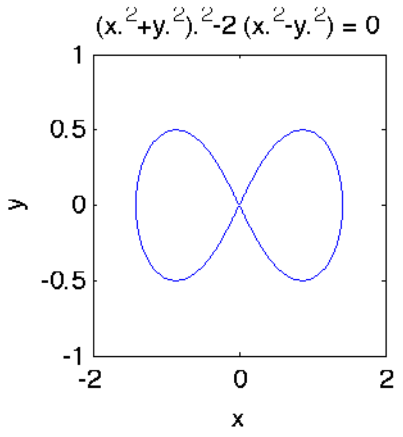


Figure 3.21. Constructing the lemniscate with an anonymous function.

Note that the result in **Figure 3.21** is identical to that found with string input in **Figure 3.20**(c).

3.2 Exercises

The general parametric form of *Lissajous curves* is given by the set of parametric equations

$$\begin{aligned} x &= a\cos(\omega_x t - \delta_x) \\ y &= b\sin(\omega_y t - \delta_y). \end{aligned} \tag{3.6}$$

There is a theorem that says that the curve will eventually repeat (or close) if the quotient ω_x/ω_y is rational. In **Exercises 1-4**, use the set of Lissajous **equations (3.6)** to sketch y versus x for the given constants. Investigate until you have a minimal domain for t that allows the curve to close.

1. $a = 2$, $b = 3$, $w_x = 2$, $w_y = 3$, $\delta_x = 1$, $\delta_y = 0$

2. $a = 5$, $b = 3$, $w_x = 4$, $w_y = 5$, $\delta_x = 0$, $\delta_y = 0$

3. $a = 2$, $b = 3$, $w_x = 3$, $w_y = 5$, $\delta_x = 0$, $\delta_y = 0$

4. $a = 5$, $b = 4$, $w_x = 4$, $w_y = 7$, $\delta_x = 1$, $\delta_y = 2$

Imagine a circle of radius b rolling about the circumference of a circle of radius a without any slippage occurring. A point P on the circumference of the outer circle of radius b traces out a curve called an *epicycloid*. The parametric equations of the epicylcoid are

$$\begin{aligned} x &= (a+b)\cos t - b\cos((1+a/b)t) \\ y &= (a+b)\sin t - b\sin((1+a/b)t). \end{aligned}$$

In **Exercises 5-8**, use Matlab to sketch the graph of the epicycloid for the given values of a and b. In each case, determine a minimal domain on t so that the curve closes.

5. $a = 8$ and $b = 5$.

6. $a = 5$ and $b = 3$.

7. In the case where $a = 2b$, a *nephroid* is produced. Set $a = 4$ and $b = 2$.

8. In the case where $a = b$, a *cardioid* is produced. Set $a = b = 2$.

9. The *Lemniscate of Bernoulli* has parametric equatons

$$\begin{aligned} x &= \frac{\cos t}{1+\sin^2 t} \\ y &= \frac{\cos t \sin t}{1+\sin^2 t}. \end{aligned}$$

Sketch the graph of the lemniscate for $0 \le t \le 2\pi$.

In **Exercises 10-13**, use Matlab's **polar** command to produce a plot of the given polar equation.

10. $r = 1 + \cos\theta$

11. $r = 1 - 2\cos\theta$

12. $r = 3 - 2\sin\theta$

13. $r = 3\sin 2\theta$

In **Exercises 14-17**, use the transformations

$$x = r\cos\theta \quad \text{and} \quad y = r\sin\theta$$

to plot the given polar equations in Cartesian coordinates.

14. $r = \cos 2\theta$

15. $r = -3\cos\theta$

16. $r = 1 - 3\cos\theta$

17. $r = 1 + 2\cos\frac{\theta}{2}$

In **Exercises 18-21**, use Matlab's **pol2cart** command to tranform the polar coordinates of the given polar equation into Cartesian coordinates. Use the **plot** command to plot the result.

18. $r = -1 - \sin\theta$

19. $r = 2 - 4\sin\theta$

20. You will need to carefully select the domain to obtain a good sketch of $r = 4/(1 + \sin\theta)$.

21. $r = e^{\theta/6}$

22. The curve defined by the polar equation

$$r = e^{\sin\theta} - 2\cos 4\theta + \sin\left(\frac{2\theta - \pi}{24}\right)$$

is called the **butterfly curve**. Sketch the polar graph on the interval $0 \le \theta \le 2\pi$ and use the result to explain the name of this curve. What happens when you extend the interval on θ, say to $[0, 12\pi]$?

23. Sketch the curve defined by the polar equation

$$r = (\sin 4\theta)^4 + \cos 3\theta.$$

This result is called the **Henri's Butterfly**, discovered by Henri Berger, a student in David Cohen's prcalculus class at UCLA in the spring of 1988.

24. Sketch the curve defined by the polar equation

$$r = \cos^2 5\theta + \sin 3\theta + 0.3.$$

This result is called the **Oscar's Butterfly**, discovered by Oscar Ramirez, a student in David Cohen's prcalculus class at UCLA in the fall of 1991.

In **Exercises 25-28**, use Matlab's **ezplot** command to plot the given algebraic curve. Use the form

ezplot(fun2,[xmin,xmax,ymin,ymax])

to provide a good viewing window for the curve. In each case, **fun2** should be entered as a string.

25. $x^4 + x^2y^2 + y^4 = x(x^2 + y^2)$, known as the "Bean Curve."

26. $(x^2 - 1)(x - 1)^2 + (y^2 - 1)^2 = 0$, known as the "Bicuspid."

27. $y^2(16 - x^2) = (x^2 + 8y - 16)^2$, known as the "Cocked Hat."

28. $y^4 - 16y^2 = x^4 - 9x^2$, known as the "Devil's Curve."

In **Exercises 29-32**, use Matlab's **ezplot** command to plot the given algebraic curve. Use the form

ezplot(fun2,[xmin,xmax,ymin,ymax])

to provide a good viewing window for the curve. In each case, **fun2** should an anonymous function.

29. $x^4 = 4(x^2 - y^2)$, known as the "Eight Curve."

30. $(x^2+y^2)(x(x+5)+y^2) = 12xy^2$, known as "Kepler's Folium."

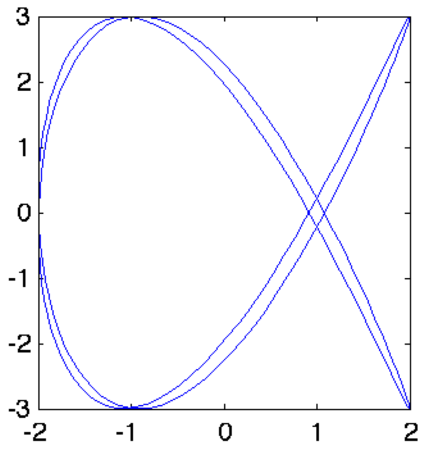
31. $16y^2 = x^3(8 - x)$, called a "Piriform."

32. $(x^2-1)^2 = y^2(3+2y)$, called the "Knot Curve."

3.2 Answers

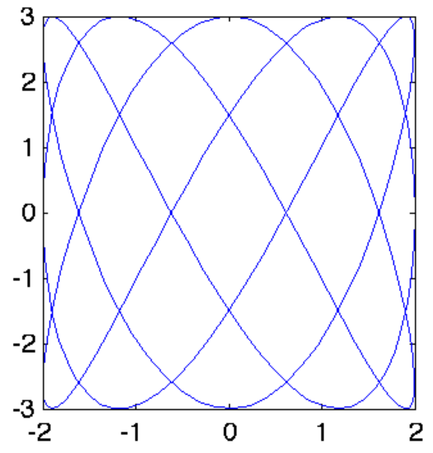
1.

```
a=2; b=3;
wx=2; wy=3;
dx=1; dy=0;
t=linspace(0,2*pi,200);
x=a*cos(wx*t+dx);
y=b*sin(wy*t+dy);
comet(x,y)
plot(x,y)
```



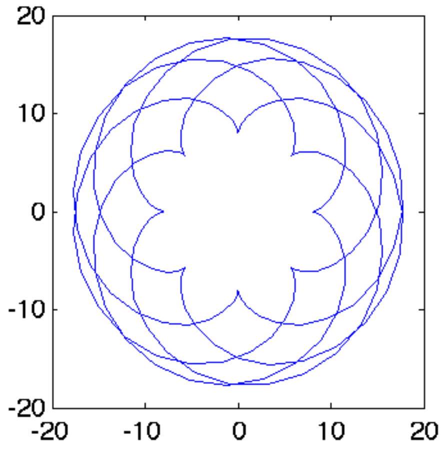
3.

```
a=2; b=3;
wx=3; wy=5;
dx=0; dy=0;
t=linspace(0,2*pi,200);
x=a*cos(wx*t+dx);
y=b*sin(wy*t+dy);
comet(x,y)
plot(x,y)
```



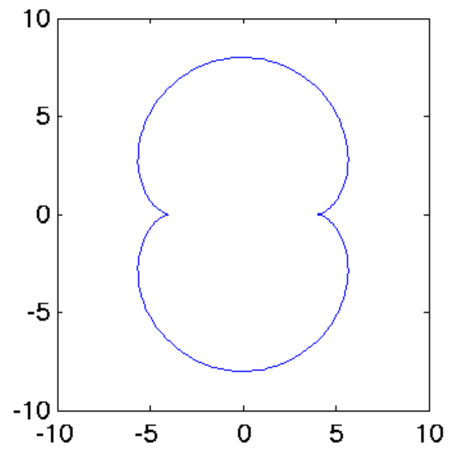
5.

```
a=8; b=5;
t=linspace(0,10*pi,200);
x=(a+b)*cos(t)-b*cos((1+a/b)*t);
y=(a+b)*sin(t)-b*sin((1+a/b)*t);
plot(x,y)
```



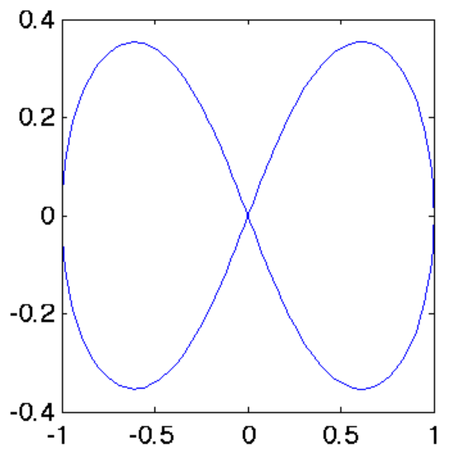
7.

```
a=4; b=2;
t=linspace(0,2*pi,200);
x=(a+b)*cos(t)-b*cos((1+a/b)*t);
y=(a+b)*sin(t)-b*sin((1+a/b)*t);
plot(x,y)
```



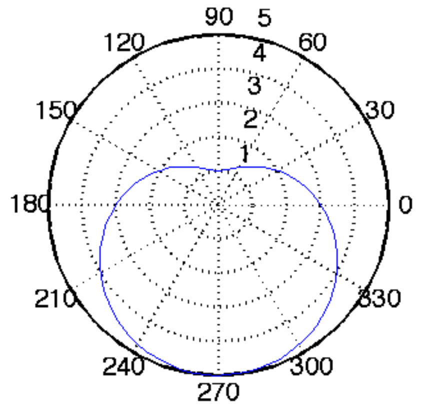
9.

```
t=linspace(0,2*pi,200);
x=cos(t)./(1+sin(t).^2);
y=cos(t).*sin(t)./(1+sin(t).^2);
plot(x,y)
```



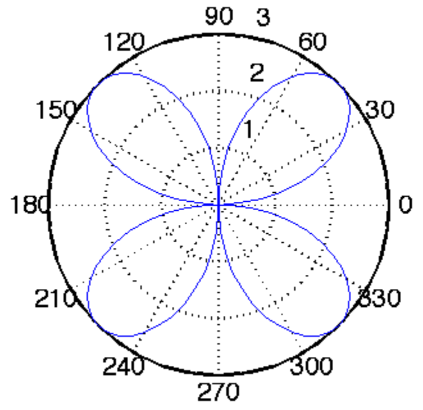
11.

```
theta=linspace(0,2*pi);
r=3-2*sin(theta);
polar(theta,r)
```



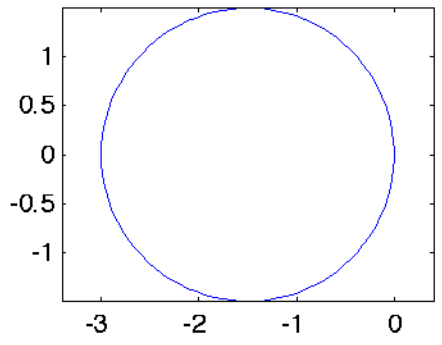
13.

```
theta=linspace(0,2*pi);
r=3*sin(2*theta);
polar(theta,r)
```



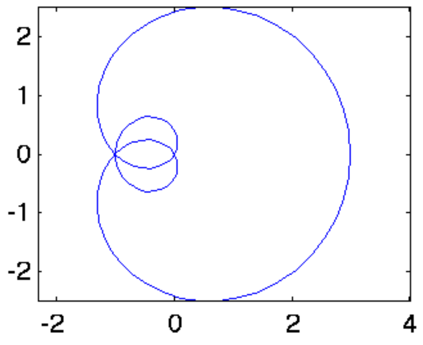
15.

```
theta=linspace(0,pi);
r=-3*cos(theta);
x=r.*cos(theta);
y=r.*sin(theta);
plot(x,y)
axis equal
```



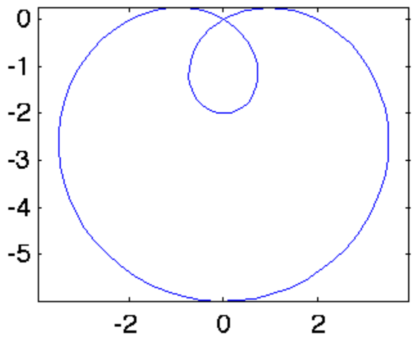
17.

```
theta=linspace(0,4*pi);
r=1+2*cos(theta/2);
x=r.*cos(theta);
y=r.*sin(theta);
plot(x,y)
axis equal
```



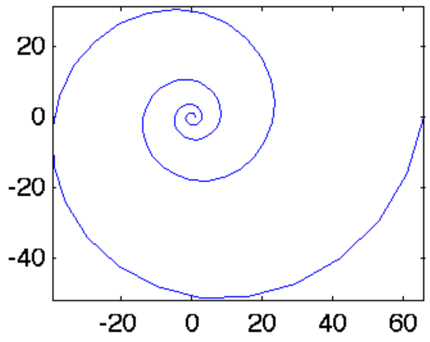
19.

```
theta=linspace(0,2*pi);
r=2-4*sin(theta);
[x,y]=pol2cart(theta,r);
plot(x,y)
axis equal
```



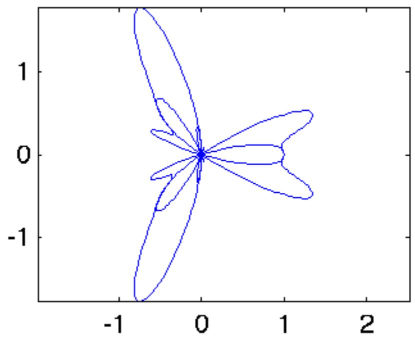
21.

```
theta=linspace(0,8*pi);
r=exp(theta/6);
[x,y]=pol2cart(theta,r);
plot(x,y)
axis equal
```



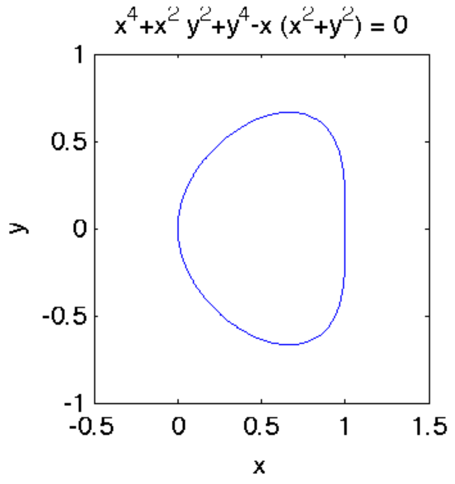
23.

```
theta=linspace(0,2*pi,500);
r=sin(4*theta).^4+cos(3*theta);
[x,y]=pol2cart(theta,r);
plot(x,y)
axis equal
```



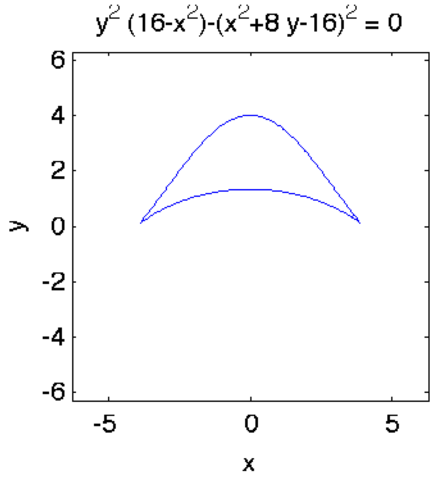
25.

```
fun2='x^4+x^2*y^2+y^4-x*(x^2+y^2)';
ezplot(fun2,[-0.5,1.5,-1,1])
```



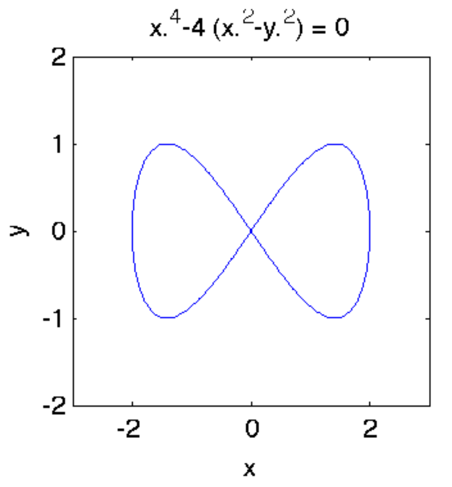
27.

```
fun2='y^2*(16-x^2)-(x^2+8*y-
16)^2';
ezplot(fun2)
```



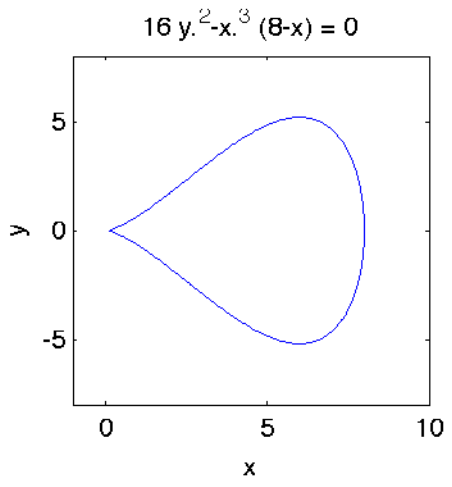
29.

```
fun2=@(x,y) x.^4-4*(x.^2-y.^2);
ezplot(fun2,[-3,3,-2,2])
```



31.

```
fun2=@(x,y) 16*y.^2-x.^3.*(8-
x);
ezplot(fun2,[-1,10,-8,8])
```



3.3 Surfaces in Matlab

In this section we add another dimension and look at the plots of surfaces in three-space. We've already spent time in two-space, drawing graphs in the Cartesian plane pictured in **Figure 3.22**(a). In the image in **Figure 3.22**(a), we use arrows and labeling to indicate the positive direction on each axis.

To add a third dimension, we need to add another axis. It is not required to make the axes orthogonal (perpendicular), but things are much easier if each pair of axes is orthogonal. So we create third axis, which we will call the z-axis, that is orthogonal to the plane containing the x- and y-axes. To determine the positive direction, we use what is known as the *right-hand rule*. If we cup the fingers of our right hand from the x- to the y-axis, then our thumb will point in the positive direction on the z-axis (out of the printed page).

It is difficult to draw three-dimensional images on a two dimensional page or screen. In **Figure 3.22**(b), note that we've made an attempt to picture the z-axis. In this image, it is our intent to deliver the message that the z-axis is perpendicular to the xy-plane and emanates from the origin directly out of the page.

As long as we maintain the right-hand rule to orient our axes, we can depict our coordinate system in a number of different ways. The orientation we prefer to use in Matlab is shown in **Figure 3.22**(c), where the positive x-axis emanates outward from the printed page. Note that if we cup the fingers of our right hand from the x-axis to the y-axis, the thumb points upward in the direction of the positive z-axis.

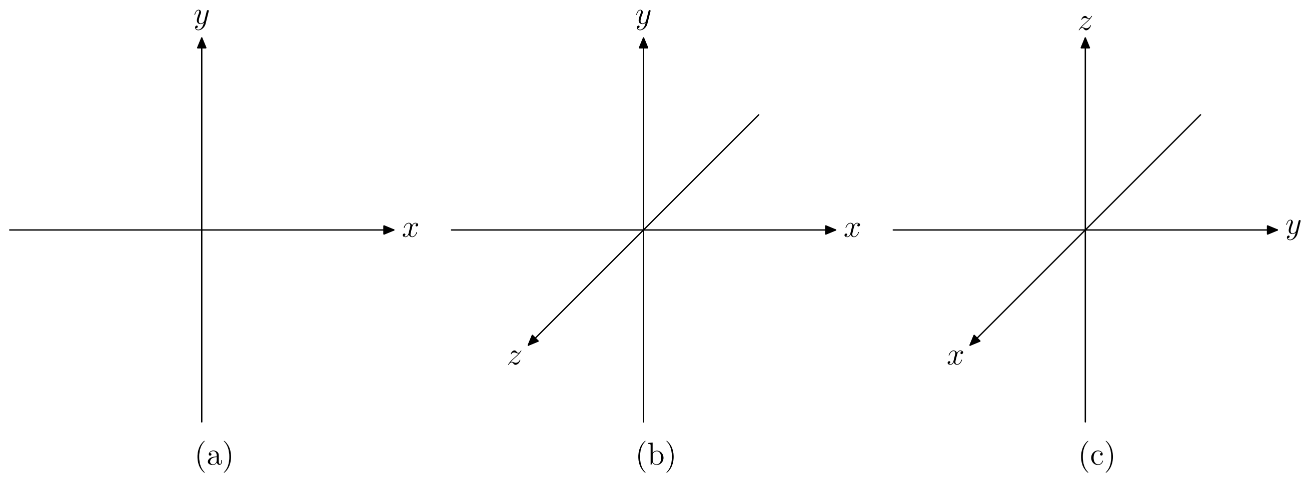


Figure 3.22. Cartesian coordinate systems obey the right-hand rule.

5 Copyrighted material. See: http://msenux.redwoods.edu/Math4Textbook/

Plotting Functions of Two Variables

We will now begin the task of plot a function of two variables. In the same way as an equation $y = f(x)$ leads to the plotting of ordered pairs (x, y) in the Cartesian plane, the equation $z = f(x, y)$ will lead to the plotting of ordered triplets (x, y, z) in three-space. Consider, for example, the function

$$f(x, y) = 12 - x - 2y.$$

If we evaluate this function at the point $(x, y) = (2, 2)$, then

$$z = f(2, 2) = 12 - 2 - 2(2) = 6.$$

This leads to the ordered triplet $(x, y, z) = (2, 2, 6)$, which we plot in three-space, as shown in **Figure 3.23**(a).

Now, suppose that we form all possible ordered pairs (x, y), where x and y are chosen from the set $\{0, 1, 2, 3\}$. We obtain a *grid* of (x, y) pairs. Next, evaluate the function $z = f(x, y) = 12 - x - 2y$ at each (x, y) pair in this grid, then plot the resulting triplets (x, y, z), as shown in **Figure 3.23**(b).

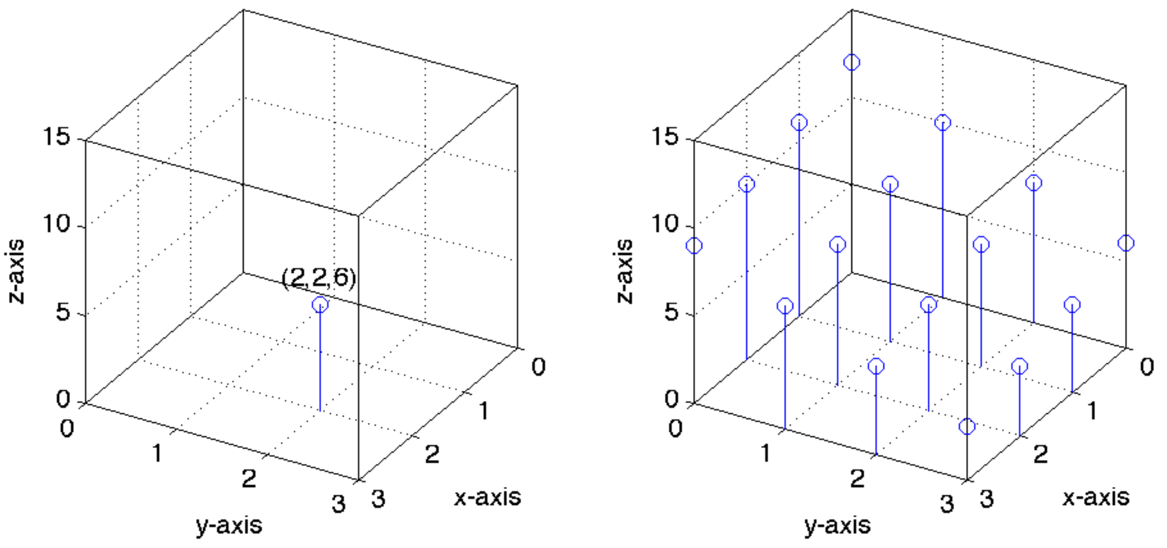


Figure 3.23. Ploting points (x, y, z) that satisfy $z = f(x, y) = 12 - x - 27$.

If we connect each of the points in **Figure 3.23**(b) with line segments (in the same way as we connect points plotted with Matlab's **plot** command), we get the surface shown in **Figure 3.24**.

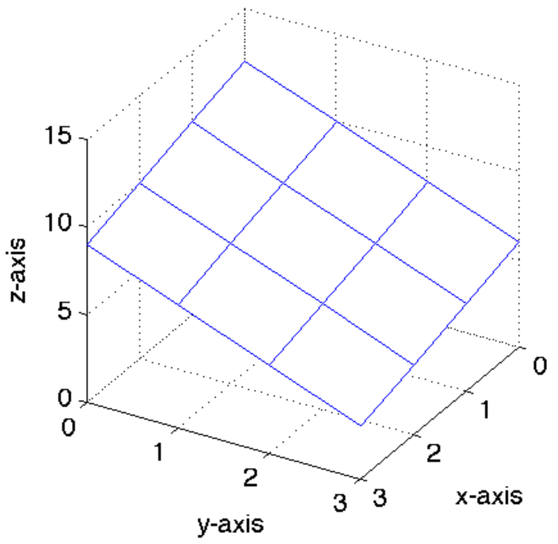


Figure 3.24. The set of points satisfying the equation $z = f(x) = 12 - x - 2y$ is a plane.

Creating the Grid. Note that in creating the image in **Figures 3.23**(b) and **3.24**, we evaluated the function $z = f(x, y) = 12 - x - 2y$ at each point of the grid shown in **Table 3.6**(a). The z-results are shown in **Table 3.6**(b).

$(0,0)$	$(1,0)$	$(2,0)$	$(3,0)$
$(0,1)$	$(1,1)$	$(2,1)$	$(3,1)$
$(0,2)$	$(1,2)$	$(2,2)$	$(3,2)$
$(0,3)$	$(1,3)$	$(2,3)$	$(3,3)$

(a)

12	11	10	9
10	9	8	7
8	7	6	5
6	5	4	3

(b)

Table 3.6. Evaluate $z = f(x, y) = 12 - x - 2y$ at each pair (x, y) in a rectangular grid.

Therefore, the question arises. How do we create a grid of (x, y) pairs in Matlab?

Because both x and y range from 0 to 3 by ones, the following command will create the grid we need.

```
>> [x,y]=meshgrid(0:3);
```

First, examine the matrix x. Note that each row of the matrix x contains the entries 0, 1, 2, and 3.

```
>> x
x =
     0     1     2     3
     0     1     2     3
     0     1     2     3
     0     1     2     3
```

Next, examine the matrix y. Note that each column contains the entries 0, 1, 2, and 3.

```
y =
     0     0     0     0
     1     1     1     1
     2     2     2     2
     3     3     3     3
```

Now, imagine superimposing the matrix y atop the matrix x. If you can hold this image in your mind, then you have a picture of the ordered pairs in **Table 3.6**(a).

It is now a simple matter to evaluate the function $z = f(x, y) = 12 - x - 2y$ at each ordered pair (x, y). Note that the following z-values agree with those found in **Table 3.6**(b).

```
>> z=12-x-2*y
z =
    12    11    10     9
    10     9     8     7
     8     7     6     5
     6     5     4     3
```

Finally, to produce the plane shown in **Figure 3.24**, execute the following commands.

```
>> mesh(x,y,z)
>> view(125,30)
>> xlabel('x-axis')
>> ylabel('y-axis')
>> zlabel('z-axis')
```

The commands **xlabel**, **ylabel**, and **zlabel** are used to label the x-, y-, and z-axes respectively. The **view** command sets the angle of the view from which an observer sees the current three-dimensional plot. The syntax is **view(ax,el)**, where **az** is the azimuth or horizontal rotation of the xy-plane, and **el** is the vertical elevation, both measured in degrees. By rotating the horizontal plane through 125 degrees, we bring it into an orientation that approximates that shown in **Figure 3.22**(c). Alternatively, the user can execute the command **mesh(x,y,z)**, then click the **Rotate 3D** icon on the toolbar, then use the mouse interactively to rotate the surface into an orientation resembling that in **Figure 3.24**.

It's not necessary that z be expressed as a function of x and y. You could have x as a function of y and z, or y as a function of x and z. Let's look at an example.

► **Example 1.** *Sketch the graph of $y = 1$ in three-space.*

Here, thought not explicit, we assume that y is a function of x and z. That is, we interpret $y = 1$ to mean $y = f(x, z) = 1$.

Because y is a function of x and z, we will first create a grid of points in the xz-plane. We arbitrarily let x run from -2 to 2 in increments of 1, and z run from -3 to 3 in increments of 1.

```
>> [x,z]=meshgrid(-2:2,-3:3);
```

It is again instructive to view the contents of matrix x. Note that each row of matrix x runs from -2 to 2 in increments of 1.

```
>> x
x =
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
```

Similarly, each column of z runs from -3 to 3 in increments of 1.

```
>> z
z =
    -3    -3    -3    -3    -3
    -2    -2    -2    -2    -2
    -1    -1    -1    -1    -1
     0     0     0     0     0
     1     1     1     1     1
     2     2     2     2     2
     3     3     3     3     3
```

It is important to note that the **meshgrid** command always creates two matrices of equal size. In this case, both x and z have dimensions 7×5.

Superimposing (mentally) matrix z atop matrix z gives a vision of (x, z) pairs. We must now evaluate the function $y = f(x, z) = 1$ at each pair. This is easy to do mentally, because the resulting y-value is always equal to 1, no matter what (x, z) pair you use. Thus, what we need Matlab to do is create a matrix of all ones having the same size as the matices x and z, and store the result in the variable y.

```
>> y=ones(size(x))
y =
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

Note that the command **y=ones(size(z))** would have worked equally well.

All that remains is to plot the surface with Matlab's **mesh** command. The following commands will produce the plane shown in **Figure 3.25**.

```
>> mesh(x,y,z)
>> axis tight
>> view(120,30)
>> xlabel('x-axis')
>> ylabel('y-axis')
>> zlabel('z-axis')
```

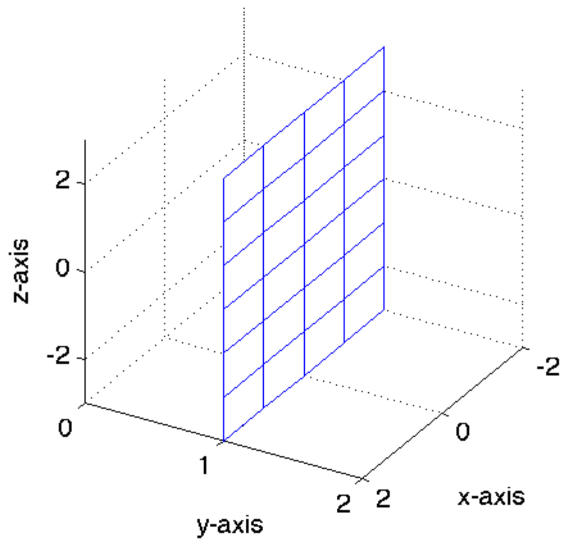


Figure 3.25. The graph of $y = 1$ in three-space.

A Bit More Interesting

We can draw a lot more interesting surfaces than simple planes with Matlab's **mesh** command. Let's look at some examples.

▶ **Example 2.** *Sketch the graph of* $z = f(x, y) = 9-x^2-y^2$ *over the rectangular domain* $D = \{(x, y) : -3 \leq x, y \leq 3\}$.

When sketching the graph of a surface that curves in space, you will again want to avoid the "Jaggies" by plotting enough points. So, let's try letting both x and y run from -3 to 3 in increments of 0.1.

```
>> [x,y]=meshgrid(-3:.1:3);
```

Evaluate the function at each point of the grid.

```
z=9-x.^2-y.^2;
```

We use of array exponentiation because we don't want to square the matrices x and y, we want to square each element of the matrices x and y. Draw the surface with the **mesh** command.

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

Finally, rotate the surface and add axis labels with the following commands. The result is shown in **Figure 3.26**.

```
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
```

By default, Matlab uses color to indicate height. You can add a colorbar to your plot with Matlab's **colorbar** command.

```
>> colorbar
```

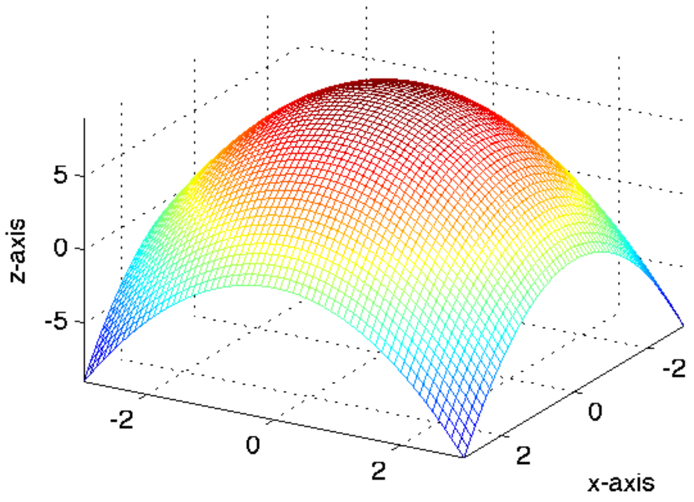


Figure 3.26. The graph of $z = f(x, y) = 9 - x^2 - y^2$ in three-space.

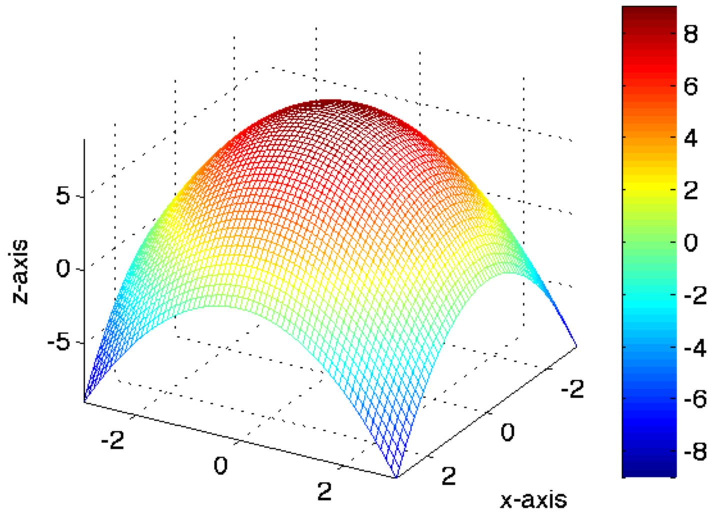


Figure 3.27. A colorbar correlates height (z-direction) and color.

In **Figure 3.27**, you see that a colorbar has been added to the figure as a result of the last command.

The colors toward the blue end of the color map indicate low heights (z-wise) and the colors at the red end of the color map indicate high heights (z-wise). Inbetween, there is a gradual gradient from blue to red. Note that the highest points on the surface are red and the lowest points on the surface are blue. Other points on the surface are colored according to their z-height and the correlation indicated by the colorbar.

Let's look at another example.

▶ **Example 3.** *Sketch the surface defined by* $z = f(x, y) = (x^2 - y^2)e^{-x^2-y^2}$ *on the domain* $D = \{(x, y) : -3 \leq x, y \leq 3\}$.

Create the grid as in the previous example.

```
>> [x,y]=meshgrid(-3:0.1:3);
```

Remember to use array operators when calculating z.

```
>> z=(x.^2-y.^2).*exp(-x.^2-y.^2);
```

The **mesh** command provides the surface shown in **Figure 3.28**.

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

Of course, you will want to label the axes and add a title.

```
>> xlabel('x-axis')
>> ylabel('y-axis')
>> zlabel('z-axis')
>> title('The graph of z = (x^2-y^2)e^{-x^2-y^2}.')
```

Sometimes you would like to add a bit of transparency to your plot, allowing the user to "see through" the mesh for details that are nomally hidden from view. Matlab's **hidden off** command provides this capability. The following command provides the image in **Figure 3.29**.

```
>> hidden off
```

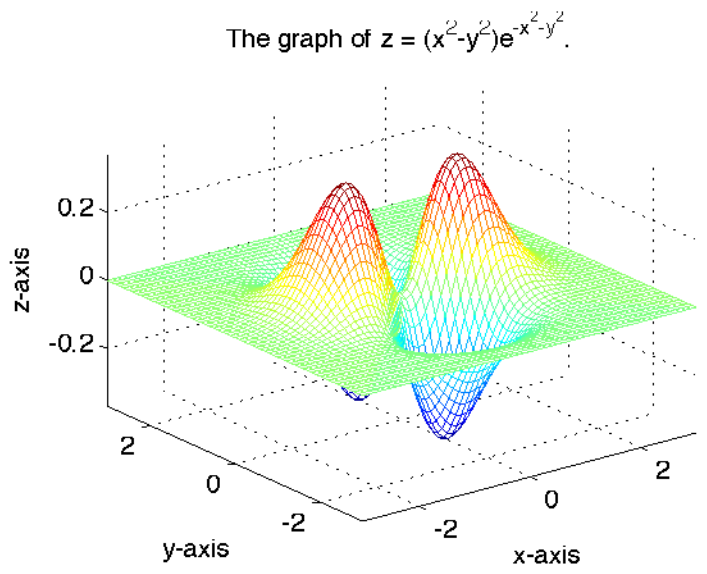


Figure 3.28. The surface defined by the function $z = f(x, y) = (x^2 - y^2)e^{-x^2-y^2}$.

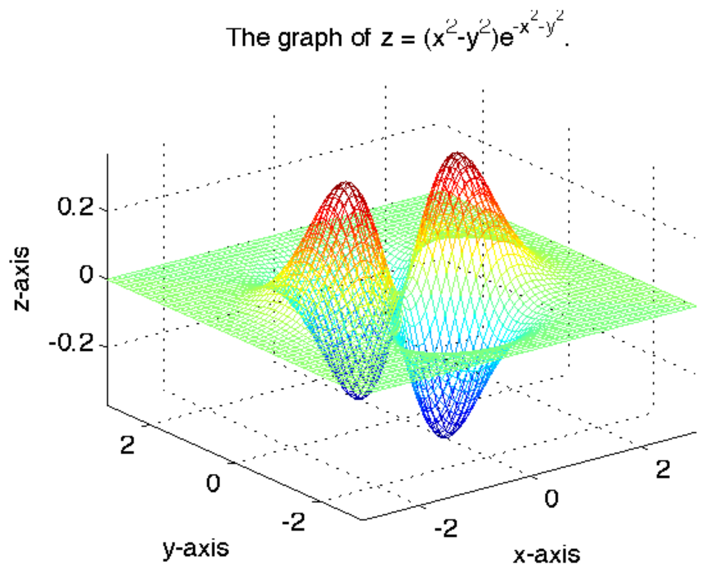


Figure 3.29. Hidden line removal allows the use to "see through" the mesh.

3.3 Exercises

In **Exercises 1-16**, use Matlab to sketch the graph of the given function on the indicated domain. Rotate each result with Matlab's **view(az,el)** command to something near the standard orientation, then add axes labels and a title containing the equation of the surface.

1. $z = f(x, y) = 10 - 2x - y$ on

$$D = \{(x, y) : -3 \le x, y \le 3\}.$$

2. $z = f(x, y) = 2$ on

$$D = \{(x, y) : -3 \le x, y \le 3\}.$$

3. $y = f(x, z) = x + 2z - 5$ on

$$D = \{(x, z) : -3 \le x, z \le 3\}.$$

4. $x = f(y, z) = y - 2z + 4$ on

$$D = \{(y, z) : -3 \le y, z \le 3\}.$$

5. $x = f(y, z) = -2$ on

$$D = \{(y, z) : -2 \le y, z \le 2\}.$$

6. $z = f(x, y) = 10 - 5x$ on

$$D = \{(x, y) : |x| \le 2, \ |y| \le 3\}.$$

7. $y = f(x, z) = -2$ on

$$D = \{(x, z) : x| \le 2, \ |z| \le 5\}.$$

8. $y = f(x, z) = 12 - 4x$ on

$$D = \{(x, z) : -1 \le x \le 4, \ |z| \le 4\}.$$

9. $z = f(x, y) = 2x - y$ on

$$D = \{(x, y) : x| \le 1, \ |y| \le 3\}.$$

10. $x = f(y, z) = 4 - z$ on

$$D = \{(y, z) : 0 \le y \le 4, \ -1 \le z \le 5\}.$$

11. $z = f(x, y) = 1/(1 + x^2 + y^2)$ on

$$D = \{(x, y) : |x| \le 2, \ |y| \le 2\}.$$

12. $z = f(x, y) = \cos(x^2 + y^2)$ on

$$D = \{(x, y) : |x| \le 2, \ |y| \le 2\}.$$

13. $z = f(x, y) = \sin \sqrt{x^2 + y^2}$ on

$$D = \{(x, y) : |x| \le 2, \ |y| \le 2\}.$$

14. $z = f(x, y) = \sin \sqrt{x^2 + y^2} / \sqrt{x^2 + y^2}$ on

$$D = \{(x, y) : |x| \le 2, \ |y| \le 2\}.$$

15. $z = f(x, y) = xe^{-x^2 - y^2}$ on

$$D = \{(x, y) : |x| \le 2, \ |y| \le 3\}.$$

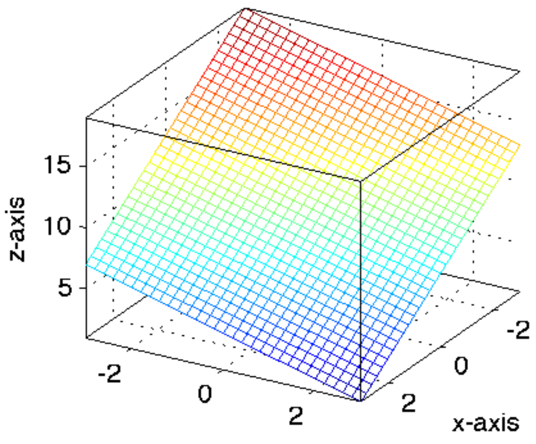
16. $z = f(x, y) = \cos x \cos y$ on

$$D = \{(x, y) : |x| \le 2\pi, \ |y| \le 2\pi\}.$$

3.3 Answers

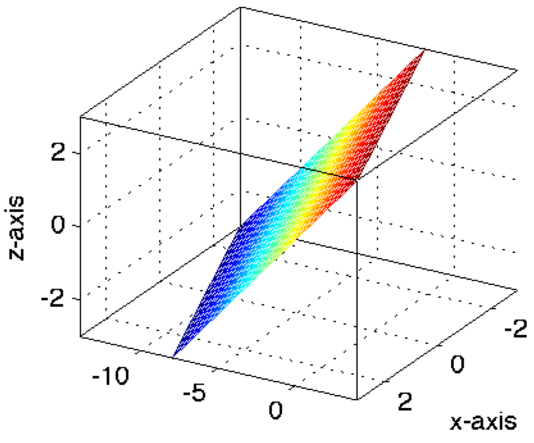
1.

```
[x,y]=meshgrid(-3:0.2:3);
z=10-2*x-y;
mesh(x,y,z)
axis tight
box on
view(120,30)
```



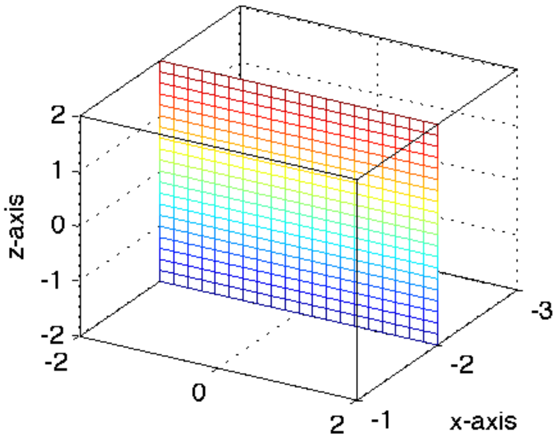
3.

```
[x,z]=meshgrid(-3:0.2:3);
y=x+2*z-5;
mesh(x,y,z)
axis tight
box on
view(120,30)
```



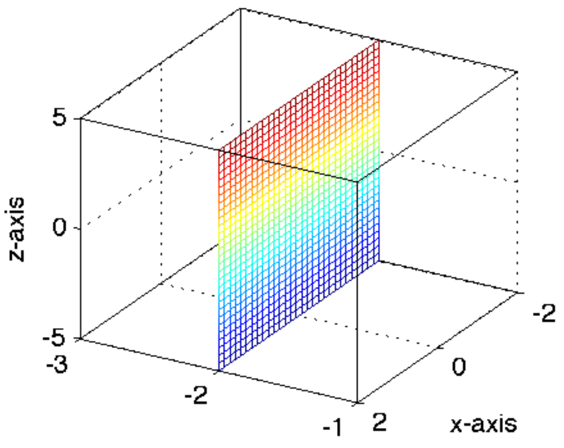
5.

```
[y,z]=meshgrid(-2:0.2:2);
x=-2*ones(size(y));
mesh(x,y,z)
axis tight
box on
view(120,30)
```



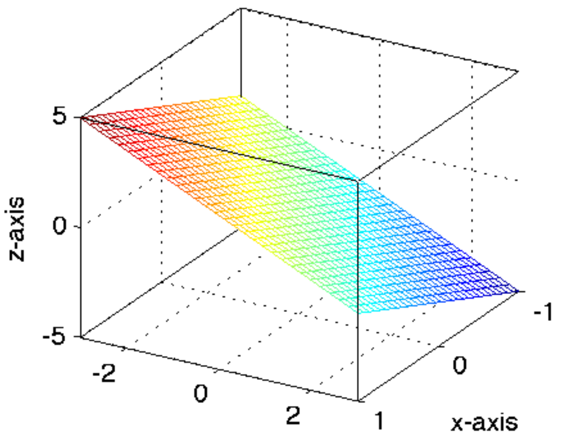
7.

```
x=linspace(-2,2,30);
z=linspace(-5,5,30);
[x,z]=meshgrid(x,z);
y=-2*ones(size(x));
mesh(x,y,z)
axis tight
box on
view(120,30)
```



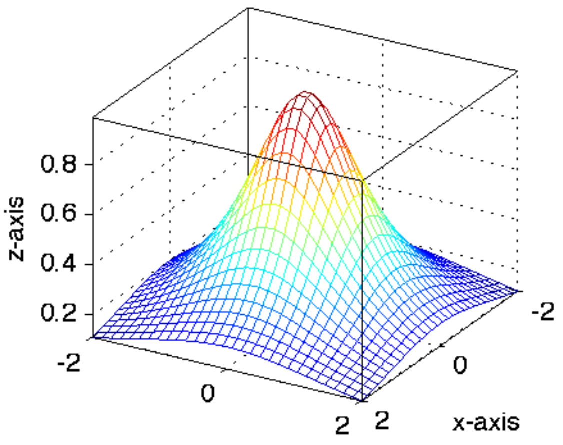
9.

```
x=linspace(-1,1,30);
y=linspace(-3,3,30);
[x,y]=meshgrid(x,y);
z=2*x-y;
mesh(x,y,z)
axis tight
box on
view(120,30)
```



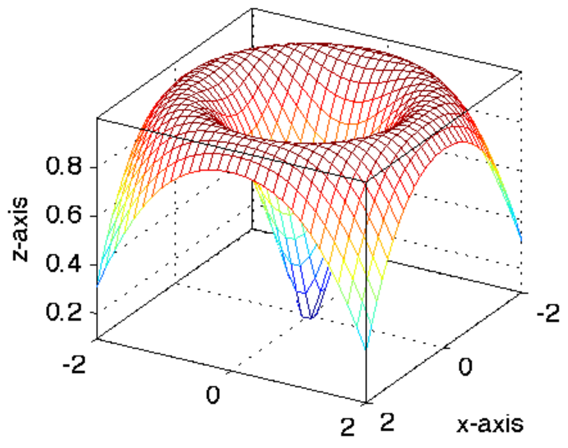
11.

```
x=linspace(-2,2,30);
y=linspace(-2,2,30);
[x,y]=meshgrid(x,y);
z=1./(1+x.^2+y.^2);
mesh(x,y,z)
axis tight
box on
view(120,30)
```



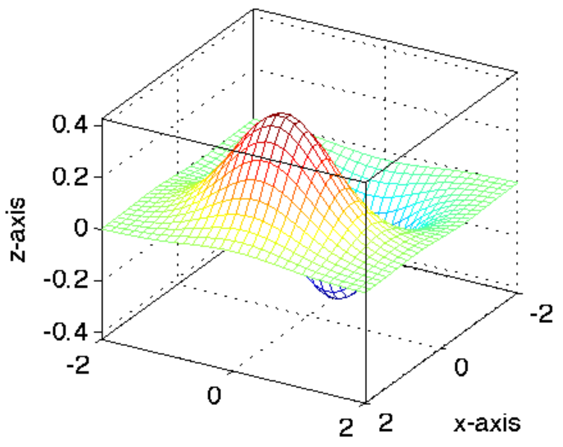
13.

```
x=linspace(-2,2,30);
y=linspace(-2,2,30);
[x,y]=meshgrid(x,y);
z=sin(sqrt(x.^2+y.^2));
mesh(x,y,z)
axis tight
box on
view(120,30)
```



15.

```
x=linspace(-2,2,30);
y=linspace(-2,2,30);
[x,y]=meshgrid(x,y);
z=x.*exp(-x.^2-y.^2);
mesh(x,y,z)
axis tight
box on
view(120,30)
```



3.4 Parametric Surfaces in Matlab

In this section we will again examine surfaces in three-space. However, the position of a point on the surface will be defined in terms of parameters. In this way we open a fascinating world of surfaces that you may never have seen before, in much the same way that parametric equatins opened a whole new world of curves in the plane.

Let's get started.

▶ **Example 1.** *Sketch the surface defined by the parametric equations*

$$\begin{aligned} x &= r\cos\theta \\ y &= r\sin\theta \\ z &= r, \end{aligned} \tag{3.7}$$

where $0 \le r \le 1$ *and* $0 \le \theta \le 2\pi$.

Note that x, y, and z are defined in terms of two parameters, r and θ, which are restriced to $0 \le r \le 1$ and $0 \le \theta \le 2\pi$. The first task is to create a grid of (r, θ) pairs, after which each pair (r, θ) will be mapped to a triplet (x, y, z) on the surface via the parametric **equations (3.7)**.

First, use the **linspace** commands to create vectors $\mathbf{r}$ and θ.

```
r=linspace(0,1,30);
theta=linspace(0,2*pi,30);
```

When using Matlab's **mesh** command, too few grid points will create a surface with the "Jaggies," but too many grid points will create a mesh that appears to be a solid mass of color, and you'll be unable to distinguish grid lines on the surface. The number 30 usually provides a nice number of grid points to start your exploration, then you can adjust upward or downward from there.

The next step is to create the grid of (r, θ) pairs with Matlab's **meshgrid** command.

```
[r,theta]=meshgrid(r,theta);
```

[6] Copyrighted material. See: http://msenux.redwoods.edu/Math4Textbook/

If you remove the semicolons in each of the previous commands, you'll note that **r** and θ were first vectors, but after the **meshgrid** command, r and θ are matrices. If your find this overwriting using the same variable distasteful, you can try something like **[R,THETA]=meshgrid(r,theta)** instead. However, we have no further use of the *vectors* **r** and θ, so we are perfectly happy overwriting r and θ with the matrix output of the **meshgrid** command.

At this point, each row of the matrix r contains the contents of the former vector **r**, and each column of the matrix θ contains the contents of the vector θ. If you mentally superimpose the matrix θ atop the matrix r, you can imagine a two dimensional grid of (r, θ) pairs. We now use the parametric **equations (3.7)** to compute the triplets (x, y, z) at each pair (r, θ) in the grid. Note that this requires the use of array operators, as one would expect.

```
x=r.*cos(theta);
y=r.*sin(theta);
z=r;
```

Each triplet (x, y, z) is a point on the surface. We can use the **mesh** command to connect neighboring points with line segments.

```
mesh(x,y,z)
```

We orient the axis, "tighten" the plot, and turn the **box on** to provided some depth to the visualization.

```
view(135,30)
axis tight
box on
```

Finally, labeling the axes helps us to visualize the orientation, as we can clearly see in **Figure 3.30**.

```
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
```

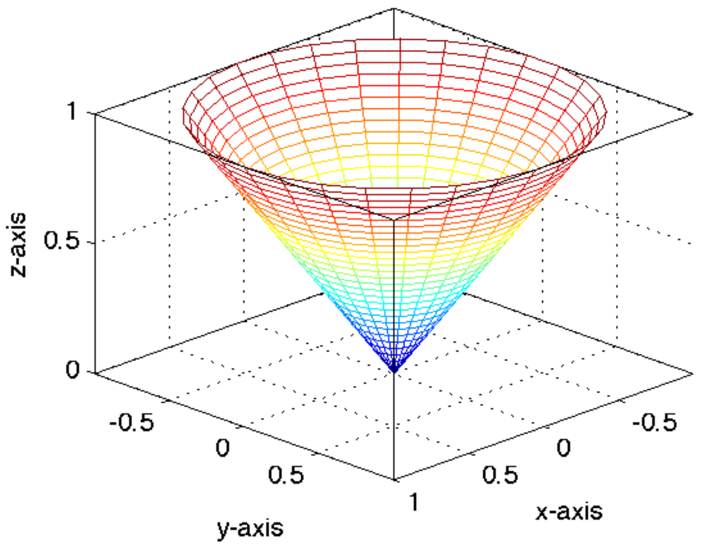


Figure 3.30. The surface defined by the parametric **equations (3.7)** is a cone.

Let's look at another example.

▶ **Example 2.** *Sketch the surface defined by the equations*

$$\begin{aligned} x &= \sin\phi\cos\theta \\ y &= \sin\phi\sin\theta \\ z &= \cos\phi, \end{aligned} \tag{3.8}$$

where $0 \le \phi \le \pi$ *and* $0 \le \theta \le 2\pi$*.*

First, create vectors ϕ and θ so that $0 \le \phi \le \pi$ and $0 \le \theta \le 2\pi$.

```
phi=linspace(0,pi,30);
theta=linspace(0,2*pi,30);
```

Now, create a grid of (ϕ, θ) pairs.

```
[phi,theta]=meshgrid(phi,theta);
```

Use the parametric **equations (3.8)** to calculate surface triplets (x, y, z) at each grid pair (ϕ, θ). Again, array operators are required.

```
x=sin(phi).*cos(theta);
y=sin(phi).*sin(theta);
z=cos(phi);
```

We can now create a mesh of the surface with Matlab's **mesh** command.

```
mesh(x,y,z)
```

We adjust the orientation, turn the **box on** to add depth of visualization, then issue **axis equal** command to show that the surface is actually a sphere, and not an ellipsoid.

```
axis equal
view(135,30)
box on
```

Finally, we annotate the axes to produce the final image in **Figure 3.31**.

```
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')T
```

Let's look at another example. It's time to have some fun!

▶ **Example 3.** *Sketch the graph of the surface defined by the parametric equations*

$$\begin{aligned} x &= 2\left[1 - e^{u/(6\pi)}\right]\cos u \cos^2\left(\frac{v}{2}\right) \\ y &= 2\left[-1 + e^{u/(6\pi)}\right]\sin u \cos^2\left(\frac{v}{2}\right) \\ z &= 1 - e^{u/(3\pi)} - \sin v + e^{u/(6\pi)}\sin v, \end{aligned} \tag{3.9}$$

where $0 \le u \le 6\pi$ *and* $0 \le v \le 2\pi$.

First, create vectors **u** and **v** so that $0 \le u \le 6\pi$ and $0 \le v \le 2\pi$.

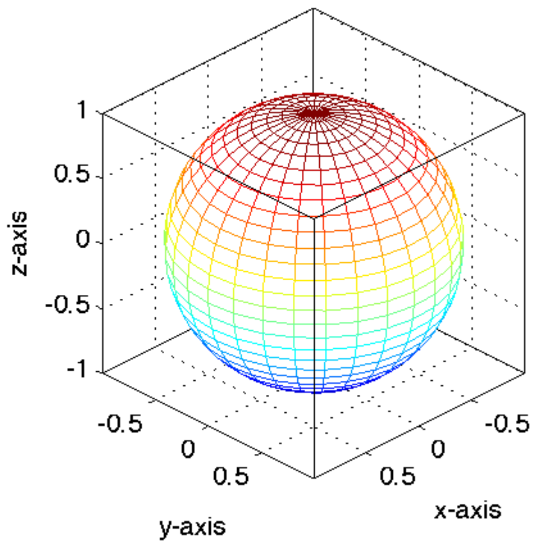


Figure 3.31. The surface defined by the parametric equations (3.8) is a sphere.

```
u=linspace(0,6*pi,60);
v=linspace(0,2*pi,60);
```

Use the vectors **u** and **v** to create a grid of (u, v) pairs.

```
[u,v]=meshgrid(u,v);
```

Use the parametric equations (3.9) to compute (x, y, z) triplets at each (u, v) pair in the grid. Again, array operators are expected.

```
x=2*(1-exp(u/(6*pi))).*cos(u).*cos(v/2).^2;
y=2*(-1+exp(u/(6*pi))).*sin(u).*cos(v/2).^2;
z=1-exp(u/(3*pi))-sin(v)+exp(u/(6*pi)).*sin(v);
```

Create a mesh of the surface with Matlab's **mesh** command.

```
mesh(x,y,z)
```

Adjust the orientation, set **axis equal**, and turn the **box on** to add depth to the visualization.

```
view(160,10)
axis equal
box on
```

Annotate the axes in the usual manner to produce the final "seashell" in **Figure 3.32**.

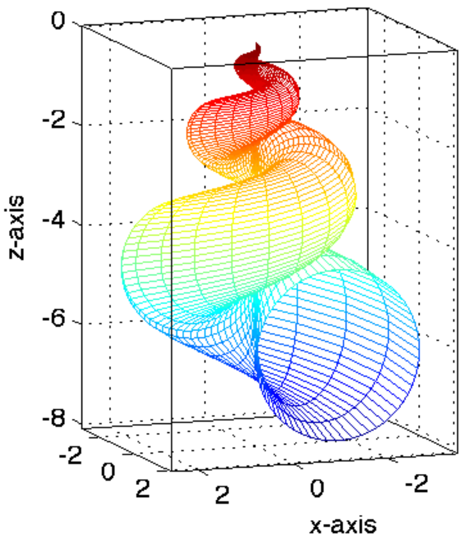


Figure 3.32. Can you hear the sound of the ocean?

Matlab also offers a **surf** command which colors the patches between the gridlines. We need only execute **surf(x,y,z)** to see the effect in **Figure 3.33**(a).

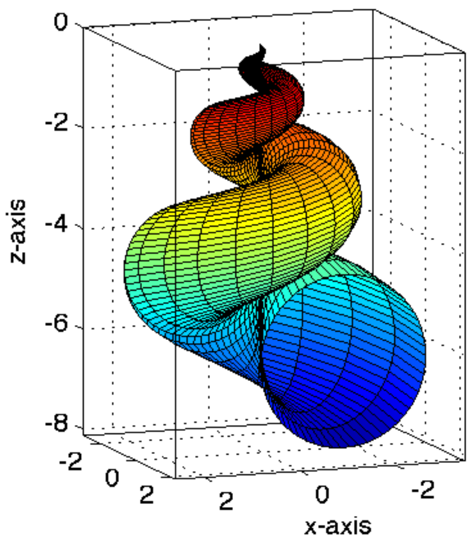
```
surf(x,y,z)
```

You might want to turn off hidden line removal and use the rotation tool on the figure toolbar to turn and twist the figure.

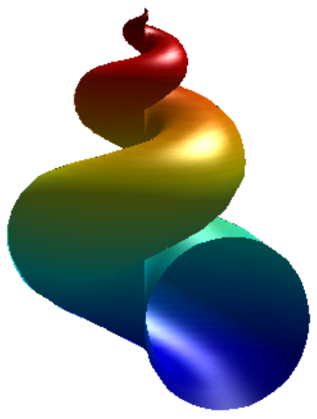
```
hidden off
```

Finally, Matlab offers truly stunning graphics capabilities. We can use the surf command in combination with edge and face coloring techniques, add lighting, and turn the axes off to produce an image in **Figure 3.33**(b) that is quite beautiful.

```
surf(x,y,z,...
    'FaceColor','interp',...
    'EdgeColor','none',...
    'FaceLighting','phong')
camlight left
view(160,10)
axis equal
axis off
```



(a)



(b)

Figure 3.33. A surface plot instead of a mesh colors the patches between meshlines on the surface. Adding lighting makes for a beautiful image.

Parametric surfaces are just too much fun! Let's look at another example.

▶ **Example 4.** *Sketch the surface defined by the parametric equations*

$$\begin{aligned} x &= \frac{v}{2}\sin\frac{u}{2} \\ y &= \left(1+\frac{v}{2}\cos\frac{u}{2}\right)\sin u \\ z &= \left(1+\frac{v}{2}\cos\frac{u}{2}\right)\cos u, \end{aligned} \tag{3.10}$$

where $0 \le u \le 2\pi$ *and* $-1 \le v \le 1$.

Create the grid with the constraints $0 \leq u \leq 2\pi$ and $-1 \leq v \leq 1$.

```
u=linspace(0,2*pi,30);
v=linspace(-1,1,15);
[u,v]=meshgrid(u,v);
```

Use the parametric **equations (3.10)** to compute surface triplete (x, y, z) at each (u, v) in the grid.

```
z=(r+v/2.*cos(u/2)).*cos(u);
y=(r+v/2.*cos(u/2)).*sin(u);
x=v/2.*sin(u/2);
```

Draw the surface.

```
surf(x,y,z)
```

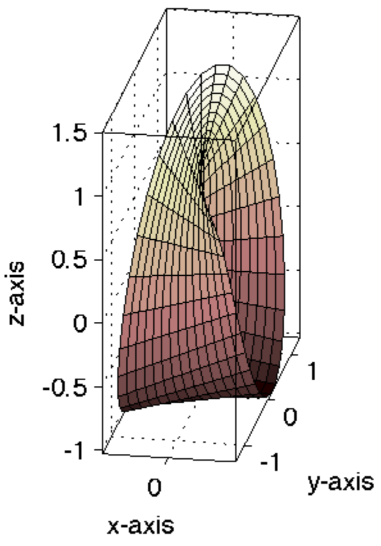
Choose the **pink** colormap[7] set the **axis equal**, and add **box on** to help with the depth of the visualization.

```
box on
axis equal
colormap pink
```

Finally, we set the orientation in order to compare our result in **Figure 3.34**(a) with the famous *Mobius Strip* drawn by Escher in **Figure 3.34**(b).

You can create a Mobius strip on your own. Simply take a long strip of paper, maybe 1-2 inches wide by 12-24 inches in length. Twist the strip once, then glue the ends together to match what you see in **Figures 3.34**(a) and (b). As the ants walk along the strip, they quickly learn that the strip has only one side, not two. The Mobius Strip is an important example that is closely studied in a subject called *differential geometry*.

[7] For information on the available colormaps in Matlab, type **help graph3d** at the command prompt.



(a)

(b)

Figure 3.34. The classic *Mobius Strip* has only one side.

Let's look at one final example, the *Klein Bottle*.

▶ **Example 5.** *Sketch the surface defined by the parametric equations*

$$\begin{aligned} x &= \left(r + \cos\frac{u}{2}\sin v - \sin\frac{u}{2}\sin 2v\right)\cos u \\ y &= \left(r + \cos\frac{u}{2}\sin v - \sin\frac{u}{2}\sin 2v\right)\sin u \qquad (3.11) \\ z &= \sin\frac{u}{2}\sin v + \cos\frac{u}{2}\sin 2v, \end{aligned}$$

with $r = 1$ *and* $0 \le u, v \le 2\pi$.

Set $r = 1$, then create the grid with constraints $0 \le u, v \le 2\pi$.

```
r=1;
u=linspace(0,2*pi,60);
v=linspace(0,2*pi,60);
[u,v]=meshgrid(u,v);
```

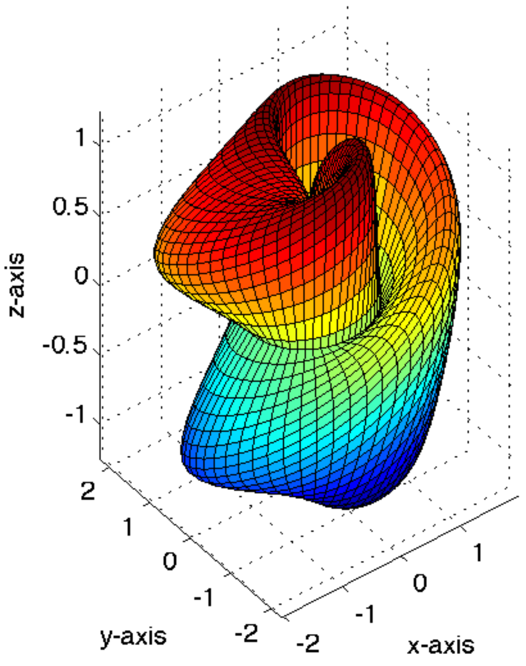
Calculate surface triplets (x, y, z) at each grid pair (u, v), using **equations (3.11)**.

```
x=(r+cos(u/2).*sin(v)-sin(u/2).*sin(2*v)).*cos(u);
y=(r+cos(u/2).*sin(v)-sin(u/2).*sin(2*v)).*sin(u);
z=sin(u/2).*sin(v)+cos(u/2).*sin(2*v);
```

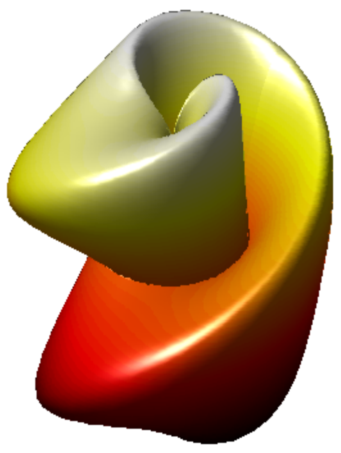
Use the **surf** command to draw the surface, a *Klein Bottle*. We'll use the default view. The result is shown in **Figure 3.35**(a).

```
surf(x,y,z)
```

Let the creative juices flow! Do some interpretation of edge and face color, add some lighting, and voila! You obtain the beautiful image shown in **Figure 3.35**(b).



(a)



(b)

Figure 3.35. A Klein Bottle, one plain, one pretty! Both fascinating!

Here is the code used to do the fancy shading and lighting seen in **Figure 3.35**(b).

```
surf(x,y,z,...
    'FaceColor','interp',...
    'EdgeColor','none',...
    'FaceLighting','phong')
camlight left
colormap(hot)
axis off
```

Note the new choice of colormap, the “hot” colormap, which adds a “heated look” to the surface of our Klein Bottle. Click the rotate icon on the toolbar, then use the mouse to twist and turn and explore this fascinating surface!

Finally, check out the hand-made Klein Bottle in **Figure 3.36** crafted by an artist.

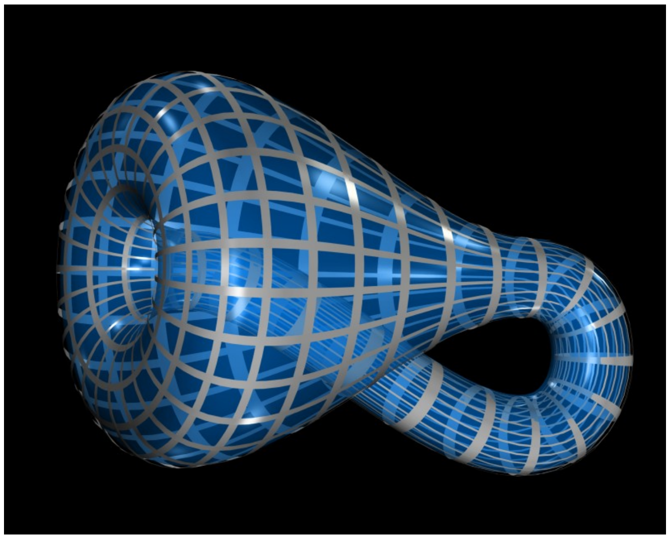


Figure 3.36. An artist’s rendering of a Klein Bottle.

3.4 Exercises

1. Sketch the *Enneper's Surface* defined by the parametric equations

$$\begin{aligned} x &= u - u^3/3 + uv^2 \\ y &= v - v^3/3 + u^2 v \\ z &= u^2 - v^2, \end{aligned}$$

where $-1.5 \leq u, v \leq 1.5$.

2. Sketch the *Ellipsoid* defined by the parametric equations

$$\begin{aligned} x &= a \cos u \sin v \\ y &= b \sin u \sin v \\ z &= c \cos v, \end{aligned}$$

where $0 \leq u \leq 2\pi$ and $0 \leq v \leq \pi$. Start with $a = 3$, $b = 4$, and $c = 5$, then experiment by varying the values of a, b, and c. Explain the effect of varying a, b, and c. You will want to use **axis equal** on this exercise.

3. Sketch the *Hyperboloid of One Sheet* defined by the parametric equations

$$\begin{aligned} x &= a \cosh(u) \cos v \\ y &= b \cosh u \sin v \\ z &= c \sinh u, \end{aligned}$$

where $-2 \leq u \leq 2$ and $0 \leq v \leq 2\pi$. Start with $a = 1$, $b = 1$, and $c = 1$, then experiment by varying the values of a, b, and c. Explain the effect of varying a, b, and c. You will want to use **axis equal** on this exercise.

4. Sketch the *Hyperboloid of One Sheet* defined by the parametric equations

$$\begin{aligned} x &= a \cosh(u) \cos v \\ y &= b \cosh u \sin v \\ z &= c \sinh u, \end{aligned}$$

where $-2 \leq u \leq 2$ and $0 \leq v \leq 2\pi$. Start with $a = 1$, $b = 1$, and $c = 1$, then experiment by varying the values of a, b, and c. Explain the effect of varying a, b, and c. You will want to use **axis equal** on this exercise.

5. Sketch the *Hyperboloid of Two Sheets* defined by the parametric equations

$$\begin{aligned} x &= a \sinh u \cos v \\ y &= b \sinh u \sin v \\ z &= c \cosh u, \end{aligned}$$

where $-2 \leq u \leq 2$ and $0 \leq v \leq 2\pi$. Start with $a = 1$, $b = 1$, and $c = 1$, then hold the graph and plot again with $a = 1$, $b = 1$, and $c = -1$ to see why this is called a *Hyperboloid of Two Sheets*.

6. Sketch the *Lissajous Surface* defined by the parametric equations

$$\begin{aligned} x &= \sin u \\ y &= \sin v \\ z &= \sin((d - au - bv)/c), \end{aligned}$$

where $-\pi \leq u, v \leq \pi$. Set $a = 1$, $b = 1$, $c = 1$ and $d = 0$. You will want to use **axis equal** on this exercise.

7. Sketch the *Whitney Umbrella* defined by the parametric equations

$$\begin{aligned} x &= uv \\ y &= u \\ z &= v^2, \end{aligned}$$

where $-1 \leq u, v \leq 1$. You will want to use **axis equal** on this exercise.

8. Sketch the *Steiner Surface* defined by the parametric equations

$$\begin{aligned} x &= \sin 2u \cos^2 v \\ y &= \sin u \sin 2v \\ z &= \cos u \sin 2v, \end{aligned}$$

where $0 \leq u \leq \pi$ and $-\pi/2 \leq v \leq \pi/2$. You will want to use **axis equal** on this exercise.

9. Sketch the *Pseudosphere* defined by the parametric equations

$$\begin{aligned} x &= \cos u \sin v \\ y &= \sin u \sin v \\ z &= \cos v + \ln(\tan(v/2)), \end{aligned}$$

where $0 \leq u \leq 2\pi$ and $0 < v < \pi$. Note that when v nears zero or π, the term $\ln(\tan(v/2)$ "leaks" to either minus or positive infinity. Keep this in mind when selecting a vector **v** to be used in an eventual grid of (u, v) pairs.

10. Sketch *Dini's Surface* defined by the parametric equations

$$\begin{aligned} x &= \cos u \sin v \\ y &= \sin u \sin v \\ z &= \cos v + \ln(\tan(v/2)) + au, \end{aligned}$$

where $0 \leq u \leq 2\pi$ and $0 < v < \pi$. Note that when v nears zero or π, the term $\ln(\tan(v/2)$ "leaks" to either minus or positive infinity. Keep this in mind when selecting a vector **v** to be used in an eventual grid of (u, v) pairs. Experiment with different values of the constant a.

11. Sketch *Helicoid* defined by the parametric equations

$$\begin{aligned} x &= av \cos u \\ y &= av \sin u \\ z &= bu, \end{aligned}$$

where $0 \leq u \leq 2\pi$ and $-d < v < d$, $d > 0$. Experiment with different values of the constants a, b, and d, then write a short description explaining how varying each of the constants a, b, and d affects the surface.

12. Sketch *Catenoid* defined by the parametric equations

$$\begin{aligned} x &= a \cos u \cosh(v/a) \\ y &= a \sin u \cosh(v/a) \\ z &= v, \end{aligned}$$

where $0 \leq u \leq 2\pi$ and $-c < v < c$, $c > 0$. Experiment with different values of the constants a and c, then write a short description explaining how varying each of the constants a and c affects the surface.

13. Sketch *Torus* defined by the parametric equations

$$\begin{aligned} x &= (a + b \cos v) \cos u \\ y &= (a + b \cos v) \sin u \\ z &= b \sin v, \end{aligned}$$

where $0 \leq u \leq 2\pi$ and $0 \leq v \leq 2\pi$. Experiment with different values of the constants a and b, then write a short description explaining how varying each of the constants a and b affects the surface.

14. Sketch the *Alien Space Ship*. Set $a = 0.4$ and define $w = \sqrt{1-a^2}$. Define

$$d = a\big[(w\cosh(au))^2 + (a\sin(wv))^2\big].$$

Now, define the parametric equations

$$\begin{aligned}
x &= -u + 2w^2\cosh(au)\sinh(au)/d\\
y &= 2w\cosh(au)\big[-w\cos v\cos(wv) - \sin v\sin(wv)\big]/d\\
z &= 2w\cosh(au)\big[-w\sin v\cos(wv) + \cos v\sin(wv)\big]/d,
\end{aligned}$$

where $-13.4 \le u \le 13.4$ and $-37.2 \le v \le 37.2$. Blast off!

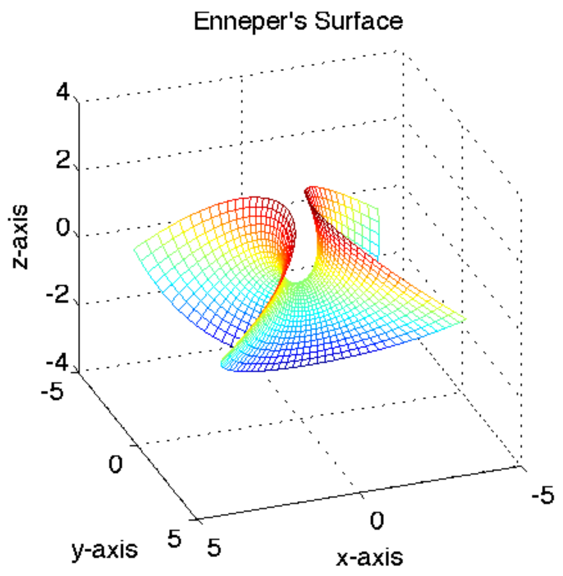
3.4 Answers

1. Create the grid and calculate **x**, **y**, and **z** at each (u, v) pair of the grid.

```
u=linspace(-1.5,1.5,40);
v=u;
[u,v]=meshgrid(u,v);
x=u-u.^3/3+u.*v.^2;
y=v-v.^3/3+u.^2.*v;
z=u.^2-v.^2;
```

Draw the surface, add some annotations, and adjust the view.

```
mesh(x,y,z)
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
title('Enneper''s Surface')
view(160,30)
```



3. Set some constants and create the grid.

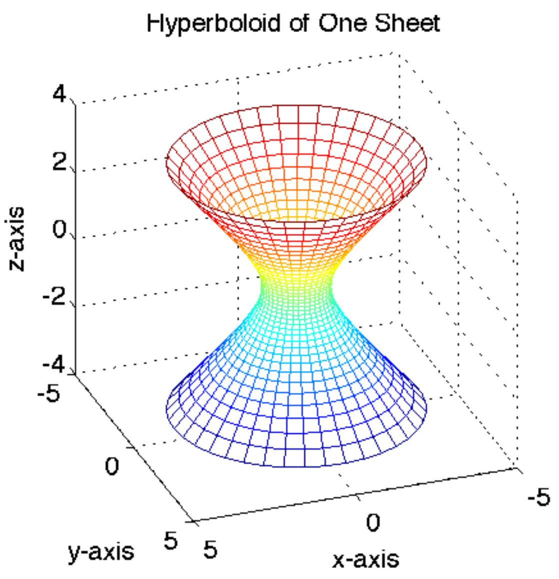
```
a=1; b=1; c=1;
u=linspace(-2,2,40);
v=linspace(0,2*pi,40);
[u,v]=meshgrid(u,v);
```

Calculate **x**, **y**,a dn **z** at each pair (u, v) in the grid.

```
x=a*cosh(u).*cos(v);
y=b*cosh(u).*sin(v);
z=c*sinh(u);
```

Draw the surface, add some annotations, and adjust the view.

```
mesh(x,y,z)
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
title('Hyperboloid of One Sheet')
view(160,30)
```



After some experimentation with values of a, b, and c, both a and b control the elliptical nature of cross sections parallel to the xy-plane, and c contols the height of the cone. The command **axis equal** will help in this exploration.

5. Set some constants and create the grid.

```
a=1; b=1; c=1;
u=linspace(-2,2,40);
v=linspace(0,2*pi,40);
[u,v]=meshgrid(u,v);
```

Calculate **x**, **y**, and **z** at each (u, v) in the grid.

```
x=a*sinh(u).*cos(v);
y=b*sinh(u).*sin(v);
z=c*cosh(u);
```

Draw the mesh.

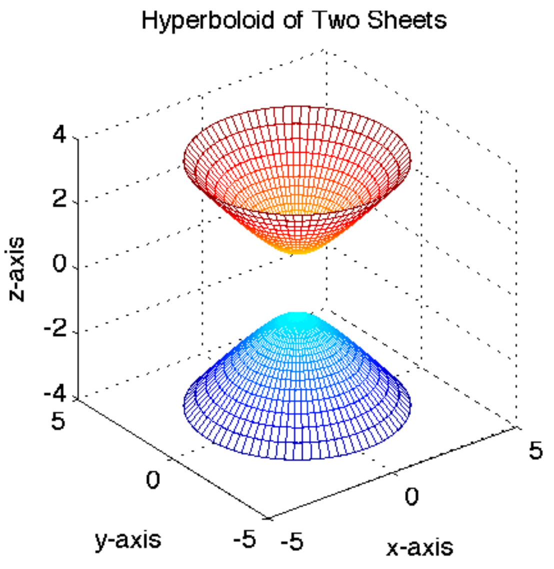
```
mesh(x,y,z)
```

Now, hold the graph, set $c = -1$, and repeat.

```
hold on
c=-1;
x=a*sinh(u).*cos(v);
y=b*sinh(u).*sin(v);
z=c*cosh(u);
mesh(x,y,z)
```

Annotate the plot.

```
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
title('Hyperboloid of Two Sheets')
```



7. Set up the grid.

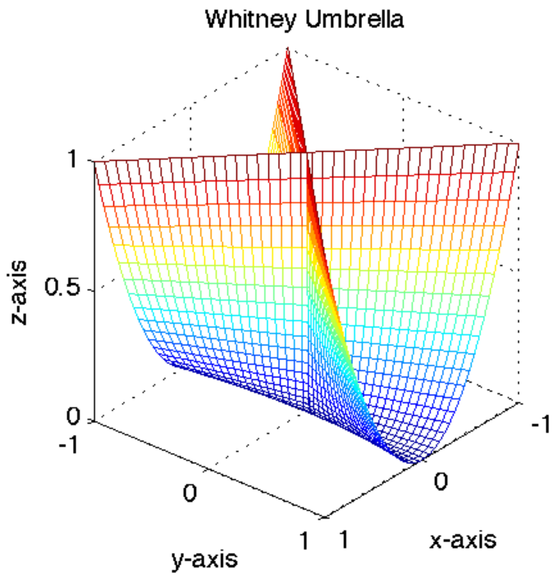
```
u=linspace(-1,1,40);
v=u;
[u,v]=meshgrid(u,v);
```

Compute **x**, **y**, and **z** at each pair (u, v) in the grid.

```
x=u.*v;
y=u;
z=v.^2;
```

Draw and orient the surface, then annotate the plot.

```
mesh(x,y,z)
view(130,30)
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
title('Whitney Umbrella')
```



9. Set up the grid. We chose to get fairly close to 0 and π in computing **v**.

```
u=linspace(0,2*pi,40);
v=linspace(pi/48,47*pi/48,40);
[u,v]=meshgrid(u,v);
```

Use the parametric equations to determine **x**, **y**, and **z** at each (u, v) pair of the grid.

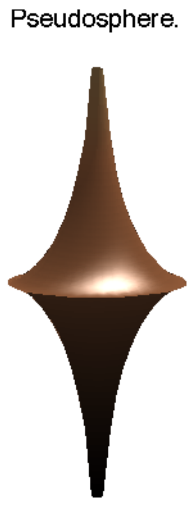
```
x=cos(u).*sin(v);
y=sin(u).*sin(v);
z=cos(v)+log(tan(v/2));
```

We use the **surf** command with some shading and lighting. For fun, we use a different colormap.

```
surf(x,y,z,...
    'FaceColor','interp',...
    'EdgeColor','none',...
    'FaceLighting','phong')
camlight left
colormap(copper)
```

We set **axis equal**, then **axis off**, orient the view, and add a title.

```
axis equal
axis off
view(160,10)
title('Pseudosphere.')
```



11. Set up some constants.

```
a=1; b=1; d=2;
```

Set up the grid.

```
u=linspace(0,2*pi,40);
v=linspace(-d,d,40);
[u,v]=meshgrid(u,v);
```

Use the parametric equations to determine **x**, **y**, and **z** at each (u, v) pair of the grid.

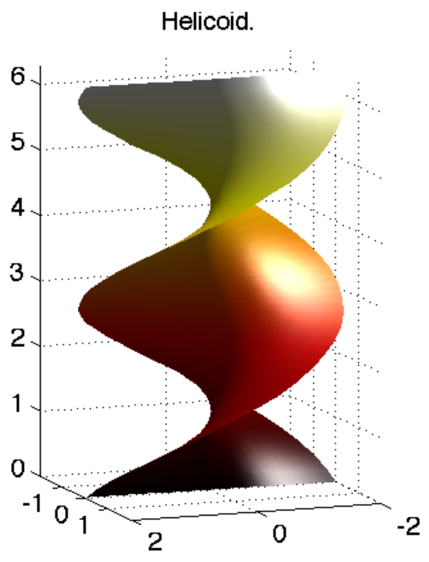
```
x=a*v.*cos(u);
y=a*v.*sin(u);
z=b*u;
```

We use the **surf** command with some shading and lighting. For fun, we use a different colormap.

```
surf(x,y,z,...
    'FaceColor','interp',...
    'EdgeColor','none',...
    'FaceLighting','phong')
camlight left
colormap(hot)
```

We set **axis equal**, orient the view, and add a title.

```
axis equal
view(160,10)
title('Helicoid.')
```



13. Set up some constants.

```
a=5; b=1;
```

Set up the grid.

```
u=linspace(0,2*pi,40);
v=u;
[u,v]=meshgrid(u,v);
```

Use the parametric equations to determine **x**, **y**, and **z** at each (u, v) pair of the grid.

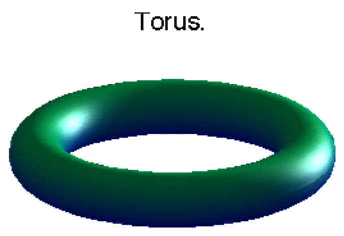
```
x=(a+b*cos(v)).*cos(u);
y=(a+b*cos(v)).*sin(u);
z=b*sin(v);
```

We use the **surf** command with some shading and lighting. For fun, we use a different colormap.

```
surf(x,y,z,...
    'FaceColor','interp',...
    'EdgeColor','none',...
    'FaceLighting','phong')
camlight left
colormap(winter)
```

We set **axis equal**, turn the axes off, orient the view, and add a title.

```
axis equal
axis off
view(150,20)
title('Torus.')
```



3.5 Space Curves in Matlab

To draw the graphs of curves in the plane, we used Matlab's **plot** command. To animate curves in the plane, we used the **comet** command. To achieve the same effects in three-space, use Matlab's **plot3** and **comet3** commands. Curves that travel through three-space are called *space curves*. Let's look at some examples.

▶ **Example 1.** *Sketch the graph of the space curve defined by the parametric equations*

$$\begin{aligned} x &= a\cos\omega t \\ y &= a\sin\omega t \\ z &= bt. \end{aligned} \tag{3.12}$$

Set $a = 2$, $b = 0.1$, $\omega = 2$, and restrict $0 \le t \le 12\pi$.

Set $a = 2$. This constant controls the amplitude of x and y. Set $b = 0.1$. As you will see, this controls the rate at which z (height) changes with respect to time. Set $\omega = 2$. This controls the rate at which the particle circles the origin, with an angular velocity of 2 radians per second.

```
a=2;
b=0.1;
w=2;
```

Create a vector of t-values using the given constraint $0 \le t \le 12\pi$.

```
t=linspace(0,12*pi,500);
```

Use the parametric **equations (3.12)** to calculate triplets (x, y, z) on the space curve in terms of t.

```
x=a*cos(w*t);
y=a*sin(w*t);
z=b*t;
```

[8] Copyrighted material. See: http://msenux.redwoods.edu/Math4Textbook/

To get a sense of the motion, use the **comet3** command.

```
comet3(x,y,z)
```

To provide a finished plot, use Matlab's **plot3** command, then add axes labels and a title. The following commands will produce the *helix* shown in **Figure 3.37**

```
plot3(x,y,z)
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
title('x = 2 cos(t), y = 2 sin(t), z = 0.1t.')
```

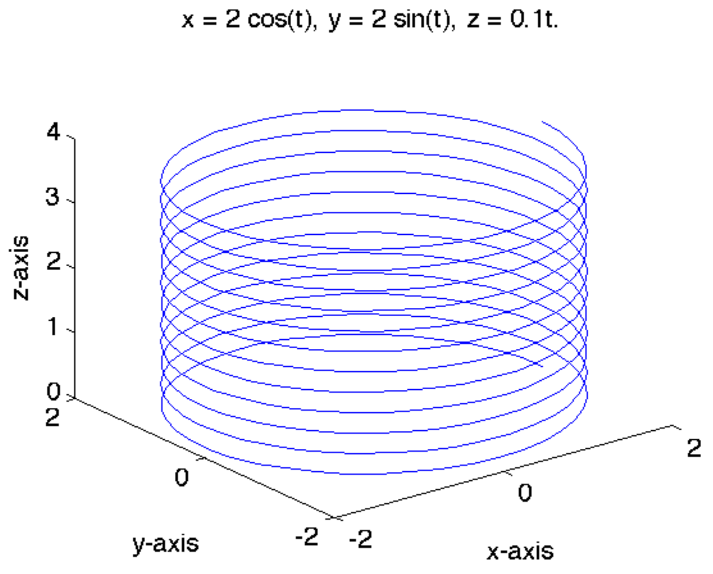


Figure 3.37. The space curve determine by the parametric **equations (3.12)** is called a *helix*.

Let's look at another example.

► **Example 2.** *Suppose that a ship travels from the south pole to the north pole keeping a fixed angle to all meridians. Then the path traveled is described by the parametric equations*

$$\begin{aligned} x &= \frac{\cos t}{\sqrt{1+\alpha^2 t^2}} \\ y &= \frac{\sin t}{\sqrt{1+\alpha^2 t^2}} \\ z &= -\frac{\alpha t}{\sqrt{1+\alpha^2 t^2}}. \end{aligned} \tag{3.13}$$

Set $\alpha = 0.2$ and restrict $-12\pi \le t \le 12\pi$.

Set $\alpha = 0.2$ and create a vector of t-values subject to the constraint $-12\pi \le t \le 12\pi$.

```
alpha=0.2;
t=linspace(-12*pi,12*pi,500);
```

Use the parametric **equations (3.13)** to compute the positions (x, y, z) on the spherical spiral as a function of time t.

```
x=cos(t)./sqrt(1+alpha^2*t.^2);
y=sin(t)./sqrt(1+alpha^2*t.^2);
z=alpha*t./sqrt(1+alpha^2*t.^2);
```

Use **comet3** to animate the motion, then follow this with Matlab's **plot3** command. This will produce the spherical spiral shown in **Figure 3.38**.

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

Handle Graphics

Many space curves are closely related with one or another particular class of surfaces. In the case of the spherical spiral, one would intuit a close relationship with a sphere. So, let's draw a sphere of appropriate radius, then superimpose the spherical spiral of **Example 2**.

In the section on Parametric Surfaces, we saw the parametric equations of a sphere of radius r, which we repeat here for convenience.

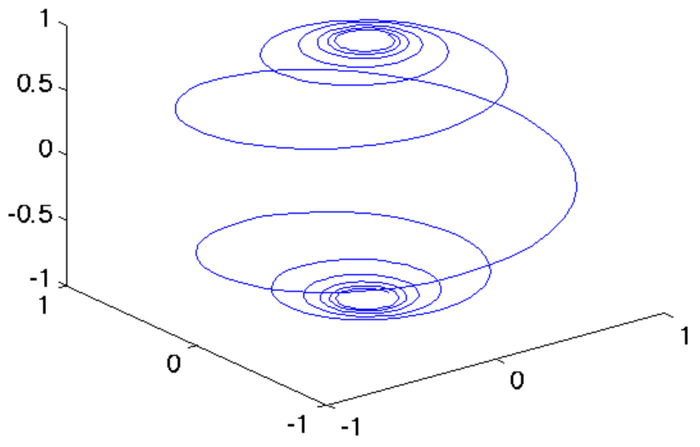


Figure 3.38. The path taken by the ship is an example of a spherical spiral.

$$\begin{aligned} x &= r\sin\phi\cos\theta \\ y &= r\sin\phi\sin\theta \\ z &= r\cos\phi \end{aligned} \tag{3.14}$$

Set $r = 1$ and create a grid of (ϕ, θ) pairs, where $0 \le \phi \le \pi$ and $0 \le \theta \le 2\pi$.

```
r=1;
phi=linspace(0,pi,30);
theta=linspace(0,2*pi,40);
[phi,theta]=meshgrid(phi,theta);
```

Use the parametric **equations (3.14)** to compute each point (x, y, z) on the surface of the sphere as a function of each grid pair (ϕ, θ).

```
x=r*sin(phi).*cos(theta);
y=r*sin(phi).*sin(theta);
z=r*cos(phi);
```

Now, plot the sphere with the **mesh** command.

```
mhndl=mesh(x,y,z)
```

The command **mhndl=mesh(x,y,z)** stores a "handle" to the mesh in the variable **mhandl**[9]. A handle is a numerical identifier associated with an object we place on the figure window. We've left the command **mhndl=mesh(x,y,z)** unsuppressed (no semicolon), so you can look in Matlab's command window to see the numerical value stored in **mhndl**.

Remember, **mhndl** is a "handle" that points at the mesh object we've just plotted. We can obtain a full list of property-value settings of this mesh by executing the command **get(mhndl)**. Matlab will respond with a huge list of property-value pairs for the current mesh. We are interested in three of these pairs: **EdgeColor**, **EdgeAlpha**, and **FaceAlpha**. We are going to set the edgecolor to a shade of gray, and we're going to make the edges and faces transparent to a certain degree. To set these property-value pairs, use Matlab's **set** command. The three consecutive dots are used by Matlab as a *line continuation character*. They indicate that you intend to continue the current command on the next line.

```
set(mhndl,...
    'EdgeColor',[0.6,0.6,0.6],...
    'EdgeAlpha',0.5,...
    'FaceAlpha',0.5)
```

If you type **get(mhndl)** at the prompt in the Matlab command window, you will see that these property-value pairs are changed to the settings we made above.

We will change the aspect ratio with the **axis equal** command, which makes the surface look truly spherical. We also turn off the axes with the **axis off** command.

```
axis equal
axis off
```

We reuse the parametric **equations (3.13)** form **Example 2** to compute points (x, y, z) on the spherical spiral as a function of t.

[9] You could use the variable **m_hndl** or **mhandle** or any variable you wish for the purpose of storing a handle to the mesh.

```
alpha=0.2;
t=linspace(-12*pi,12*pi,500);
x=cos(t)./sqrt(1+alpha^2*t.^2);
y=sin(t)./sqrt(1+alpha^2*t.^2);
z=alpha*t./sqrt(1+alpha^2*t.^2);
```

Instead of using the **plot3** command, we will use the **line** command. The **line** command is used to append graphics to the plot without erasing what is already there. When you use the **line** command, there is no need to use the **hold on** command.

```
lhndl=line(x,y,z)
```

Look in Matlab's command window to see that a numerical value has been assigned to the variable **lhndl**. This is a numerical identifier to the spherical spiral just plotted. Use **get(lhndl)** to obtain a list of property-value settings for the spherical spiral. We are interested in two of these pairs: **Color** and **LineWidth**, which we will now change with Matlab's **set** command.

```
set(lhndl,...
    'Color',[0.625,0,0],...
    'LineWidth',2)
```

These commands change the spherical spiral to a dark shade of red and thicken the spiral to 2pts. The result is shown in **Figure 3.39**. Because we changed the Alpha settings (transparency) of the edges and faces of the sphere, note that we can "see through" the sphere to a certain extent, making the spherical spiral on the far side of the sphere visible.

Viviani's Curve

Many new curves can be fromed from the intersection of two surfaces. For example, all of the conic sections (circle, ellipse, parabola, and hyperbola) are determined by how a plane intersects a right-circular cone (we will explore these conic sections in the exercises).

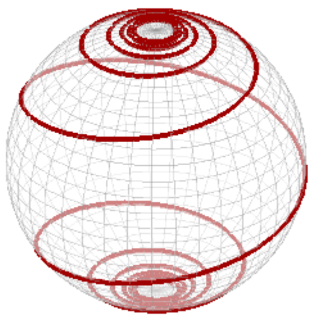


Figure 3.39. Superimposing the spherical spiral on a transparent sphere.

Such is the case with a space curve known as *Viviani's Curve*, which is the intersection of a sphere of radius $2r$ (centered at the origin) and a right-circular cylinder of radius r that is shifted r units along either the x- or y-axis.

The equation of the sphere is $x^2 + y^2 + z^2 = 4r^2$. We know that this sphere can be produced by the parametric equations

$$\begin{aligned} x &= 2r\sin\phi\cos\theta \\ y &= 2r\sin\phi\sin\theta \\ z &= 2r\cos\phi. \end{aligned} \tag{3.15}$$

We offer the following construction with few comments. It is similar to what we did in **Example 2**.

```
r=1;
phi=linspace(0,pi,30);
theta=linspace(0,2*pi,40);
[phi,theta]=meshgrid(phi,theta);

x=2*r*sin(phi).*cos(theta);
y=2*r*sin(phi).*sin(theta);
z=2*r*cos(phi);
```

Handle graphics are employed to color edges.

```
mhndl1=mesh(x,y,z)
set(mhndl1,...
    'EdgeColor',[0.6,0.6,0.6])
axis equal
axis off
```

A circle of radius r centered at the origin has equation $x^2 + y^2 = r^2$. If we plot the set of all (x, y, z) such that $x^2 + y^2 = r^2$, the result is a right-circular cylinder of radius r. Replace x with $x - r$ to get $(x - r)^2 + y^2 = r^2$, which will shift the cylinder r units in the x-direction. One final question remains. How can we parametrize the cylinder $(x - r)^2 + y^2 = r^2$?

It's fairly straightforward to show that the parametric equations

$$\begin{aligned} x &= r\cos t \\ y &= r\sin t \end{aligned} \tag{3.16}$$

produce a circle of radius r centered at the origin[10]. This can be verified with Matlab's **comet** or **plot** command[11]. To shift this r units in the x-direction add r to the equation for x to obtain

$$\begin{aligned} x &= r + r\cos t \\ y &= r\sin t. \end{aligned}$$

Thus, the parametric equations of the right-circular cylinder $(x - r)^2 + y^2 = r^2$ are

$$\begin{aligned} x &= r + r\cos t \\ y &= r\sin t \\ z &= z. \end{aligned}$$

The key to plotting the cylinder in Matlab is to realize that x, y, and z are functions of both t and z. That is,

$$\begin{aligned} x(t, z) &= r + r\cos t \\ y(t, z) &= r\sin t \\ z(t, z) &= z. \end{aligned} \tag{3.17}$$

Therefore, the first task is to create a grid of (t, z) pairs.

It will suffice to let $0 \le t \le 2\pi$. That should trace out the circle. If we hope to see the intersection of the sphere (radius $2r$) and the cylinder, we will need to

[10] $x^2 + y^2 = r^2\cos^2 t + r^2\sin^2 t = r^2(\cos^2 t + \sin^2 t) = r^2$.

[11] And **axis equal**.

have the cylinder at least as low and high in the z-directions as is the sphere of radius $2r$. Thus, limit $-2r \le z \le 2r$. After creating these vectors, we then create a grid of (t, z) pairs.

```
t=linspace(0,2*pi,40);
z=linspace(-2*r,2*r,20);
[t,z]=meshgrid(t,z);
```

Use the parametric **equations (3.17)** to produce points (x, y, z) on the cylinder as a function of the grid pairs (t, z).

```
x=r+r*cos(t);
y=r*sin(t);
z=z;
```

Hold the graph and plot the cylinder. Handle graphics are used to color the edges. A view is set that allows the visualization of the intersection of the sphere and cylinder. The resulting image is shown in **Figure 3.40**.

```
hold on
mhndl2=mesh(x,y,z)
set(mhndl2,...
    'EdgeColor',[0.8,0,0])
view(50,20)
```

Now, how do we get the parametrization of the curve of intersection? Recall the equations of the sphere and cylinder.

$$x^2 + y^2 + z^2 = 4r^2$$
$$(x - r)^2 + y^2 = r^2$$

If we expand and simplify the second equation, we get

$$x^2 - 2rx + y^2 = 0.$$

If we subtract this result from the first equation, we obtain

$$z^2 + 2rx = 4r^2.$$

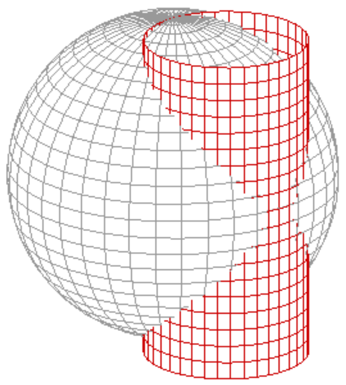


Figure 3.40. The intersection of the sphere and cylinder is called *Viviani's Curve*.

Note that all points on Viviani's curve must fall on the cylinder, where $x = r + r\cos t$. Substitute this into the last result.

$$\begin{aligned} z^2 + 2r(r + r\cos t) &= 4r^2 \\ z^2 + 2r^2 + 2r^2\cos t &= 4r^2 \\ z^2 &= 2r^2 - 2r^2\cos t. \end{aligned}$$

This can be written

$$z^2 = 4r^2\left(\frac{1-\cos t}{2}\right),$$

and the half angle identity $\sin^2(t/2) = (1-\cos t)/2$ leads to

$$z^2 = 4r^2\sin^2(t/2).$$

Normally, we should now say $z = \pm 2r\sin(t/2)$, but we will go with $z = 2r\sin(t/2)$ in the following set of parametric equations for Viviani's Curve.

$$\begin{aligned} x &= r + r\cos t \\ y &= r\sin t \\ z &= 2r\sin(t/2). \end{aligned} \tag{3.18}$$

Note that the period of $z = 2r\sin(t/2)$ is $T = 4\pi$, so if we go with only $0 \leq t \leq 2\pi$, we will only get positive values of z and the lower half of the curve will not be shown[12]. Thus, we use $0 \leq t \leq 4\pi$.

```
t=linspace(0,4*pi,200);
```

Use the parametric **equation (3.18)** to compute points (x, y, z) on Viviani's Curve in terms of t.

```
x=r+r*cos(t);
y=r*sin(t);
z=2*r*sin(t/2);
```

We plot the curve and record its "handle." We then use *handle graphics* to color the curve black ($[0, 0, 0]$ is black) and set the line width at 2 points.

```
vhndl=line(x,y,z)
set(vhndl,...
    'Color',[0,0,0],...
    'LineWidth',2)
view(50,20)
```

Setting the same view used in **Figure 3.40** produces the image in **Figure 3.41**.

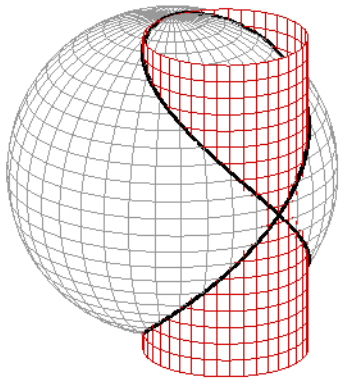


Figure 3.41. The intersection of the cylinder and sphere is the curve of Viviani.

[12] The reader should verify this statement is true.

You'll want to click the *rotate icon* on the toolbar and use the mouse to rotate and twist the figure to verify that our parametrization of Viviani's Curve is truly the intersection of the sphere and cylinder.

3.5 Exercises

In **Exercises 1-6**, perform each of the following tasks.

i. Sketch the space curve defined by the given set of parametric equations over the indicated domain.
ii. Turn the **box on**, label each axis, and provide a title.

1. On $0 \le t \le 1$, sketch the *Lissajous Curve*

$$\begin{aligned} x &= 3\sin(4\pi t)\\ y &= 4\sin(6\pi t)\\ z &= 5\sin(8\pi t). \end{aligned}$$

2. On $0 \le t \le 1$, sketch the *Lissajous Curve*

$$\begin{aligned} x &= 3\sin(4\pi t)\\ y &= 4\sin(10\pi t)\\ z &= 5\sin(14\pi t). \end{aligned}$$

3. On $0 \le t \le 2\pi$, sketch the *Torus Knot*

$$\begin{aligned} x &= (6.25 + 3\cos 5t)\cos 2t\\ y &= (6.25 + 3\cos 5t)\sin 2t\\ z &= 3.25\sin 5t. \end{aligned}$$

4. On $0 \le t \le 2\pi$, sketch the *Torus Knot*

$$\begin{aligned} x &= (7 + 2\cos 5t)\cos 3t\\ y &= (7 + 2\cos 5t)\sin 3t\\ z &= 3\sin 5t. \end{aligned}$$

5. On $0 \le t \le 6\pi$, sketch the *Trefoil Knot*

$$\begin{aligned} x &= \cos t(2 - \cos(2t/3))\\ y &= \sin t(2 - \cos(2t/3))\\ z &= -\sin(2t/3). \end{aligned}$$

6. On $0 \le t \le 10\pi$, sketch the *Cinquefoil Knot*

$$\begin{aligned} x &= \cos t(2 - \cos(2t/5))\\ y &= \sin t(2 - \cos(2t/5))\\ z &= -\sin(2t/5). \end{aligned}$$

In **Exercises 7-10**, we investigate the *conic sections*, each of which is the intersection of a plane with a right circular cone. In each exercise, perform the following tasks.

i. Draw the right circular cone having the parametric equations

$$\begin{aligned} x &= r\cos\theta\\ y &= r\sin\theta \qquad (3.19)\\ z &= r, \end{aligned}$$

where $0 \le \theta \le 2\pi$ and $-1 \le r \le 1$. Use handle graphics to set the **EdgeColor** to a shade of gray.
ii. Execute **hold on** to "hold" the surface plot of the right circular cone. Superimpose the plot of the given plane over the given domain, then use Matlab's handle graphics to set the **EdgeColor** to a single color of your choice.
iii. Click the rotate icon on the figure toolbar and rotate the figure to a view that emphasizes the conic section (curve of intersection) and note the azimuth and elevation. Use

these components in the **view(ax,el)** in your script to obtain an identical view. Label the axes and provide a title that includes the name the conic section that is the intersection of the given plane and the right circular cone.

7. Sketch the plane $z = 1/2$ over the domain

$$D = \{(x, y) : -1 \leq x, y \leq 1\}.$$

8. Sketch the plane $z = 0.4y + 0.5$ over the domain

$$D = \{(x, y) : -1 \leq x, y \leq 1\}.$$

9. Sketch the plane $z = y + 0.25$ over the domain

$$D = \{(x, y) : -1 \leq x, y \leq 1\}.$$

10. Sketch the plane $x = 0.5$ over the domain

$$D = \{(y, z) : -1 \leq y, z \leq 1\}.$$

---

11. Sketch the torus defined by the parametric equations

$$\begin{aligned} x &= (7 + 2\cos u)\cos v \\ y &= (7 + 2\cos u)\sin v \\ z &= 3\sin u. \end{aligned}$$

Set the **EdgeColor** to a shade of gray and add transparency by setting both **FaceAlpha** and **EdgeAlpha** equal to 0.5. Set the **axis equal**. Use Matlab's **line** command to superimpose the torus knot having parmetric equations

$$\begin{aligned} x &= (7 + 2\cos 5t)\cos 2t \\ y &= (7 + 2\cos 5t)\sin 2t \\ z &= 3\sin 5t \end{aligned}$$

over the time domain $0 \leq t \leq 2\pi$. Use handle graphics to set the **Color** of the torus knot to a color of your choice and set the **LineWidth** to a thickness of 2 points.

12. Sketch the torus defined by the parametric equations

$$\begin{aligned} x &= (8 + 2\cos u)\cos v \\ y &= (8 + 2\cos u)\sin v \\ z &= 3\sin u. \end{aligned}$$

Set the **EdgeColor** to a shade of gray and add transparency by setting both **FaceAlpha** and **EdgeAlpha** equal to 0.5. Set the **axis equal**. Use Matlab's **line** command to superimpose the torus knot having parmetric equations

$$\begin{aligned} x &= (8 + 2\cos 11t)\cos 3t \\ y &= (8 + 2\cos 11t)\sin 3t \\ z &= 3\sin 11t \end{aligned}$$

over the time domain $0 \leq t \leq 2\pi$. Use handle graphics to set the **Color** of the torus knot to a color of your choice and set the **LineWidth** to a thickness of 2 points.

13. Sketch the cone defined by the parametric equations

$$\begin{aligned} x &= r\cos\theta \\ y &= r\sin\theta \\ z &= r \end{aligned}$$

where $0 \leq r \leq 1$ and $0 \leq \theta \leq 2\pi$. Set the **EdgeColor** to a shade of gray and add transparency by setting both **FaceAlpha** and **EdgeAlpha** equal to 0.5. Set the **axis equal**. Use Matlab's **line** command to superimpose the *conical spiral* having parmetric equations

$$\begin{aligned} x &= t\cos 20t \\ y &= t\sin 20t \\ z &= t \end{aligned}$$

over the time domain $0 \le t \le 1$. Use handle graphics to set the **Color** of the conical spiral to a color of your choice and set the **LineWidth** to a thickness of 2 points.

14. Sketch the cylinder defined by the parametric equations

$$\begin{aligned} x &= 2\cos\theta \\ y &= 2\sin\theta \\ z &= z, \end{aligned}$$

where $0 \le z \le 5$ and $0 \le \theta \le 2\pi$. Set the **EdgeColor** to a shade of gray and add transparency by setting both **FaceAlpha** and **EdgeAlpha** equal to 0.5. Set the **axis equal**. Use Matlab's **line** command to superimpose the helix having parmetric equations

$$\begin{aligned} x &= 2\cos 5t \\ y &= 2\sin 5t \\ z &= t, \end{aligned}$$

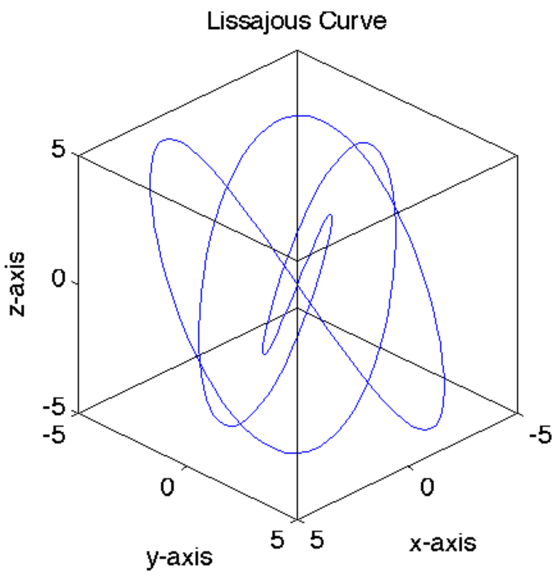
over the time domain $0 \le t \le 5$. Use handle graphics to set the **Color** of the helix to a color of your choice and set the **LineWidth** to a thickness of 2 points.

15. **Challenge**. The intersection of a sphere and a plane that passes through the center of the sphere is called a *great circle*. Sketch the equation of a sphere of radius 1, then sketch the plane $y = x$ and show that the intersection is a great circle. Find the parametric equations of this great circle and add it to your plot.

3.5 Answers

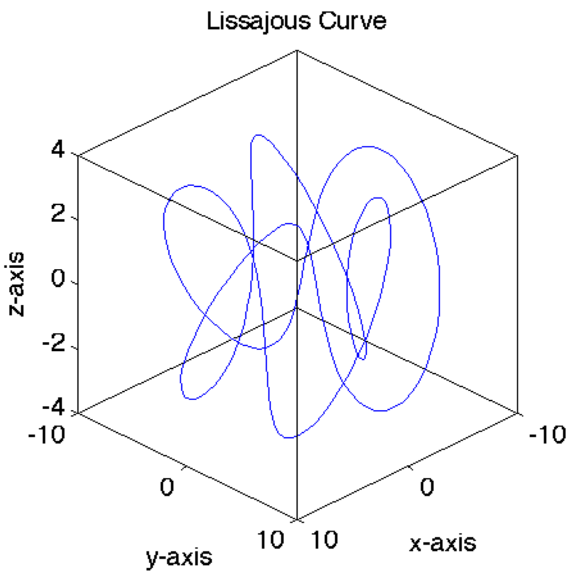
1.

```
t=linspace(0,1,200);
x=3*sin(4*pi*t);
y=4*sin(6*pi*t);
z=5*sin(8*pi*t);
plot3(x,y,z)
box on
view(135,30)
```



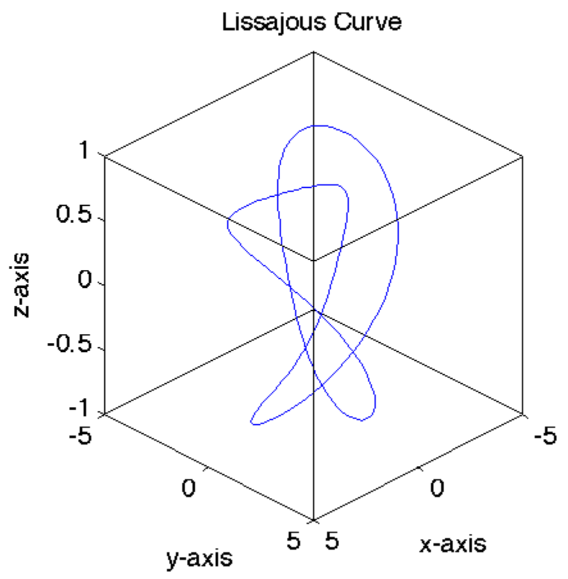
3.

```
t=linspace(0,2*pi,200);
x=(6.25+3*cos(5*t)).*cos(2*t);
y=(6.25+3\cos(5*t)).*sin(2*t);
z=3.25*sin(5*t);
plot3(x,y,z)
box on
view(135,30)
```



5.

```
t=linspace(0,6*pi,200);
x=cos(t).*(2-cos(2*t/3));
y=sin(t).*(2-cos(2*t/3));
z=-sin(2*t/3);
plot3(x,y,z)
box on
view(135,30)
```



7. Set the grid for the cone.

```
theta=linspace(0,2*pi,40);
r=linspace(-1,1,30);
[theta,r]=meshgrid(theta,r);
```

Use the parametric equations to compute x, y, and z.

```
x=r.*cos(theta);
y=r.*sin(theta);
z=r;
```

Draw the right circular cone in a shade of gray.

```
mhndl=mesh(x,y,z)
set(mhndl,...
    'EdgeColor',[.6,.6,.6])
```

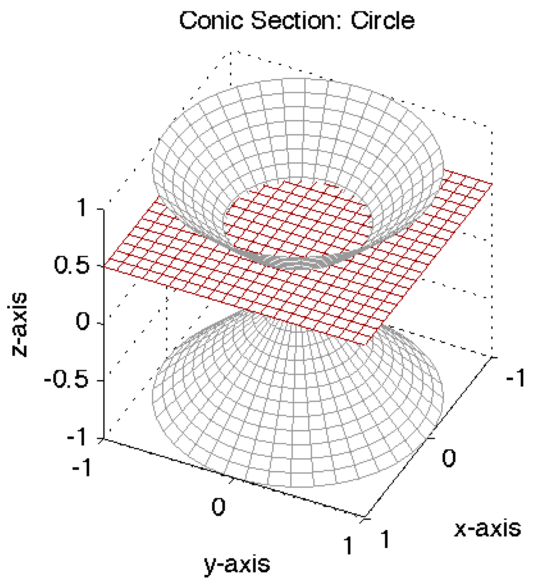
Hold the surface plot and draw the plane $z = 1/2$.

```
hold on
[x,y]=meshgrid(-1:0.2:1);
z=0.5*ones(size(x));
phndl=mesh(x,y,z);
set(phndl,...
    'EdgeColor',[0.625,0,0])
```

Adjust orientation.

```
axis equal
view(116,38)
```

Annotate the plot in the usual manner. The intersection is a circle, as indicated in the title.



9. Set the grid for the cone.

```
theta=linspace(0,2*pi,40);
r=linspace(-1,1,30);
[theta,r]=meshgrid(theta,r);
```

Use the parametric equations to

compute x, y, and z.

```
x=r.*cos(theta);
y=r.*sin(theta);
z=r;
```

Draw the right circular cone in a shade of gray.

```
mhndl=mesh(x,y,z)
set(mhndl,...
    'EdgeColor',[.6,.6,.6])
```

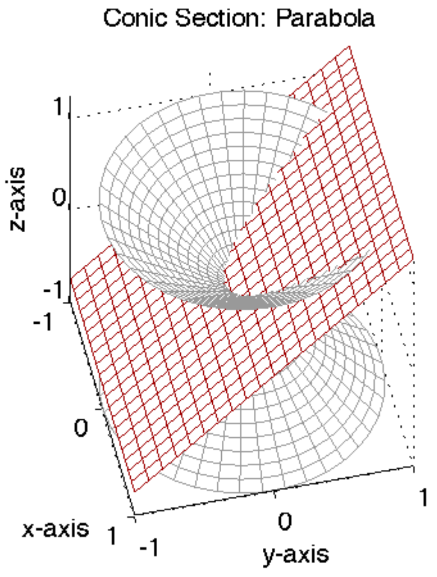
Hold the surface plot and draw the plane $z = y + 0.25$.

```
hold on
[[x,y]=meshgrid(-1:0.1:1);
z=y+0.25;
phndl=mesh(x,y,z);
set(phndl,...
    'EdgeColor',[0.625,0,0])
```

Adjust orientation.

```
axis equal
view(77,50)
```

Annotate the plot in the usual manner. The intersection is a parabola, as indicated in the title.



11. Set parameters a, b, and c.

```
a=7;b=2;c=3;
```

Set the grid for the torus.

```
u=linspace(0,2*pi,20);
v=linspace(0,2*pi,40);
[u,v]=meshgrid(u,v);
```

Use the parametric equations to compute x, y, and z.

```
x=(a+b*cos(u)).*cos(v);
y=(a+b*cos(u)).*sin(v);
z=c*sin(u);
```

Draw the torus in a shade of gray and add transparency. Set the perspective with **axis equal**.

```
mhndl=mesh(x,y,z);
set(mhndl,...
    'EdgeColor',[.6,.6,.6],...
    'FaceAlpha',0.5,...
    'EdgeAlpha',0.5);
axis equal
```

Compute x, y, and z for the torus knot over the requested time domain.

```
t=linspace(0,2*pi,200);
x=(a+b*cos(5*t)).*cos(2*t);
y=(a+b*cos(5*t)).*sin(2*t);
z=c*sin(5*t);
```

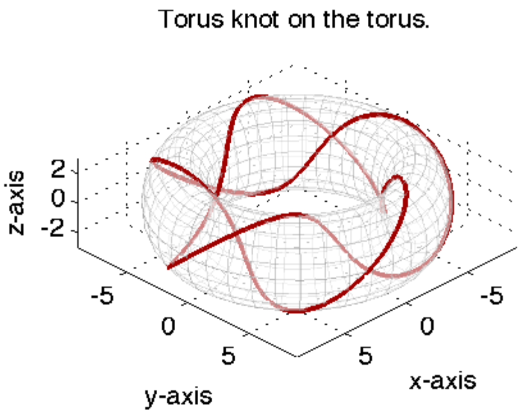
Plot the torus knot and change its color and linewidth.

```
lhndl=line(x,y,z);
set(lhndl,...
    'Color',[.625,0,0],...
    'LineWidth',2)
```

Adjust orientation.

```
view(135,30)
```

Annotate the plot in the usual manner.



13. Set the grid for the cone.

```
r=linspace(0,1,20);
theta=linspace(0,2*pi,40);
[r,theta]=meshgrid(r,theta);
```

Use the parametric equations to compute x, y, and z.

```
x=r.*cos(theta);
y=r.*sin(theta);
z=r;
```

Draw the cone in a shade of gray and add transparency. Set the perspective with **axis equal**.

```
mhndl=mesh(x,y,z);
set(mhndl,...
    'EdgeColor',[.6,.6,.6],...
    'FaceAlpha',0.5,...
    'EdgeAlpha',0.5);
axis equal
```

Compute x, y, and z for the conical spiral over the requested time domain.

```
t=linspace(0,1,200);
x=t.*cos(20*t);
y=t.*sin(20*t);
z=t;
```

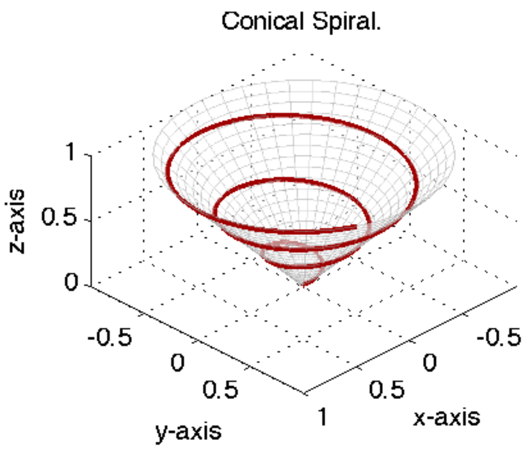
Plot the concical spiral and change its color and linewidth.

```
lhndl=line(x,y,z);
set(lhndl,...
    'Color',[.625,0,0],...
    'LineWidth',2)
```

Adjust orientation.

```
view(135,30)
```

Annotate the plot in the usual manner.


Conical Spiral.
z-axis
y-axis
x-axis