

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,'*')
```

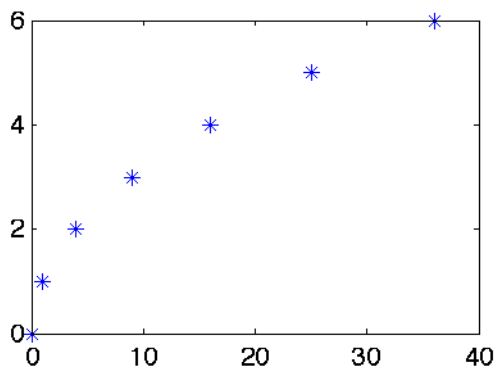


Figure 3.1. Plotting $y = \sqrt{x}$ at discrete values of x .

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, you 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 \mathbf{x} .

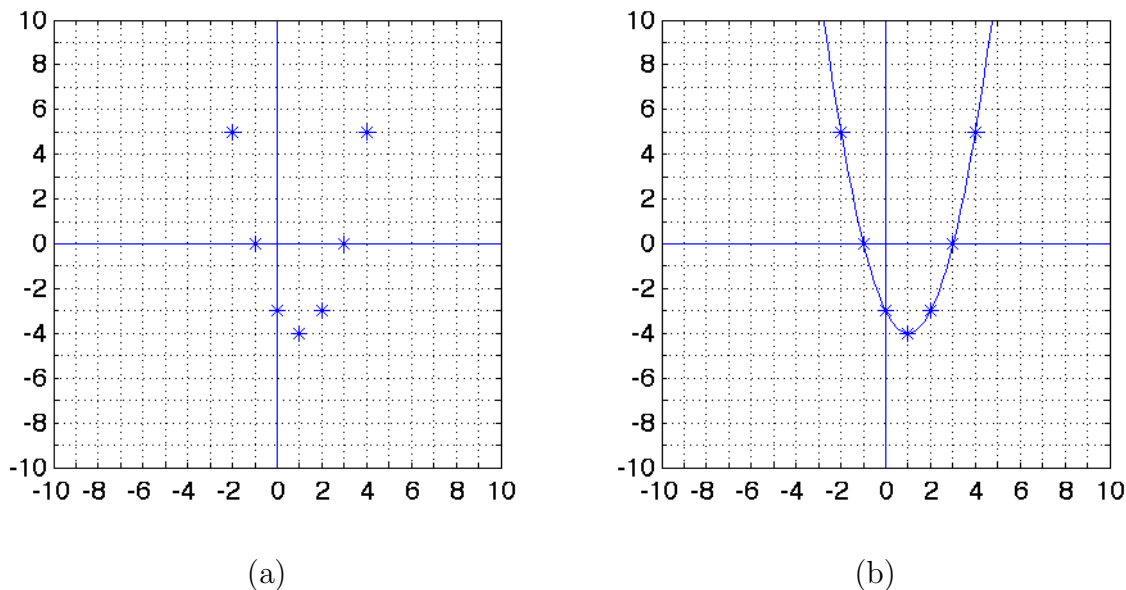


Figure 3.2. Plotting the graph of $y = x^2 - 2x - 3$.

```
>> 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 $\mathbf{y}=\mathbf{x}.\wedge 2-2*\mathbf{x}-3$ 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.

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

The array operation $.\wedge 2$ will raise each element in the vector $\mathbf{x} = [-2, -1, \dots, 4]^T$ to the second power. The product of the scalar 2 and the vector $\mathbf{x} = [-2, -1, \dots, 4]^T$ 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 \mathbf{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 \mathbf{x} , you must recompute the vector \mathbf{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,'*')
```

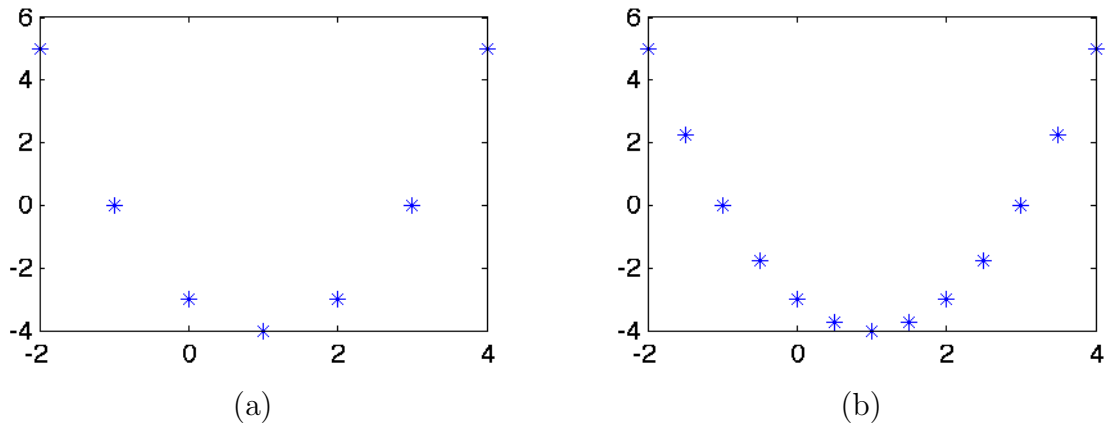


Figure 3.3. Using Matlab to plot the graph of $y = x^2 - 2x - 3$.

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 \mathbf{x} that contains a lot of points. Don’t forget to recalculate the vector \mathbf{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-')
```

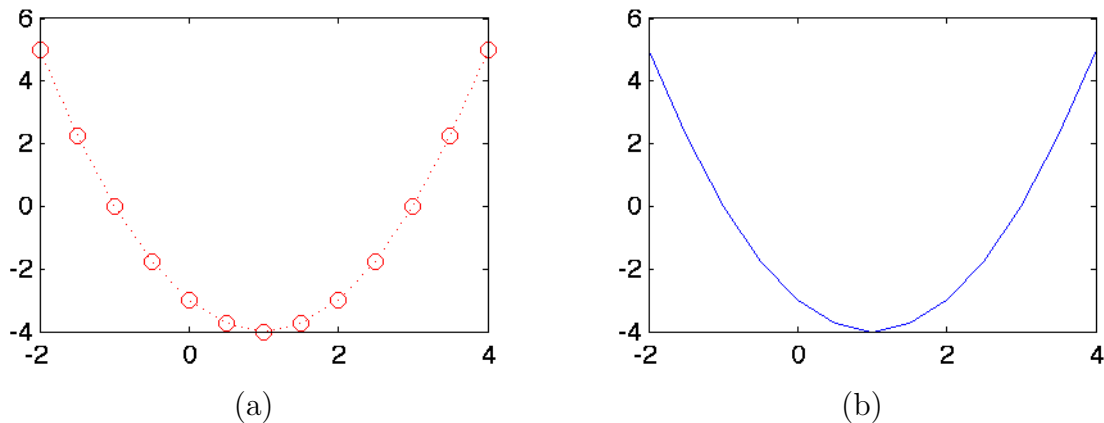


Figure 3.4. Using different plot styles with Matlab.

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 segments), 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	$(0, 0)$
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 \mathbf{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 \mathbf{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 multiplication. Hence, the operator **.*** is used.

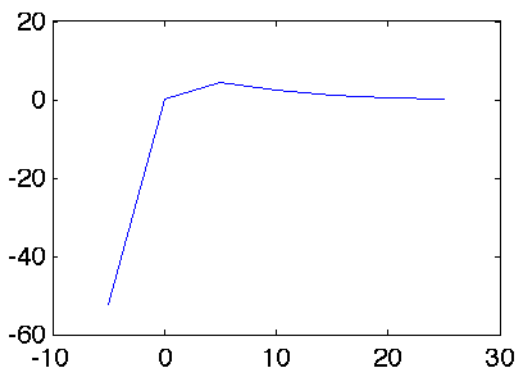
$$\mathbf{y} = \begin{bmatrix} 3(-5) \exp(-0.25(-5)) \\ 3(0) \exp(-0.25(0)) \\ \vdots \\ 3(25) \exp(-0.25(25)) \end{bmatrix}$$

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 **y** identical to the second column of **Table 3.3(b)**.

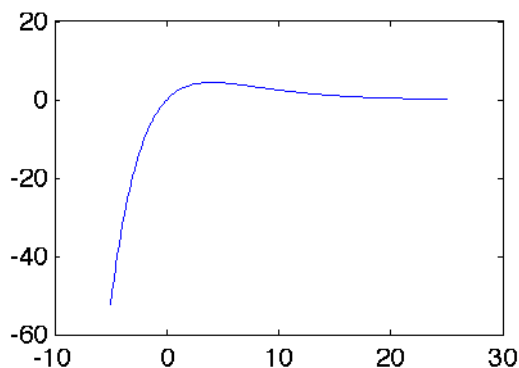
```
>> y=3*x.*exp(-0.25*x)
y =
-52.3551
     0
  4.2976
  2.4625
  1.0583
  0.4043
  0.1448
```

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's because we didn't plot enough points to emulate a smooth curve. However, this is easily rectified by adding more values to the vector \mathbf{x} and recomputing the vector \mathbf{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 $\mathbf{3*x.*exp(-0.25*x)}$, 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 $\mathbf{3*x}$ and $\mathbf{-0.25*x}$, 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 $\mathbf{exp(-0.25*x)}$ causes Matlab to take the exponential of each element of the vector $\mathbf{-0.25x}$.
3. Finally, each of the expressions $\mathbf{3*x}$ and $\mathbf{exp(-0.25*x)}$ 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	$y = 1/(1 + 99e^{-0.5t})$	t	y	(t, y)
0	$y = 1/(1 + 99e^{-0.5(0)})$	0	0.0100	(0, 0.0100)
10	$y = 1/(1 + 99e^{-0.5(10)})$	10	0.5999	(10, 0.5999)
20	$y = 1/(1 + 99e^{-0.5(20)})$	20	0.9955	(20, 0.9955)
30	$y = 1/(1 + 99e^{-0.5(30)})$	30	1.0000	(30, 1.0000)

(a)

(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))`.

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

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)
```

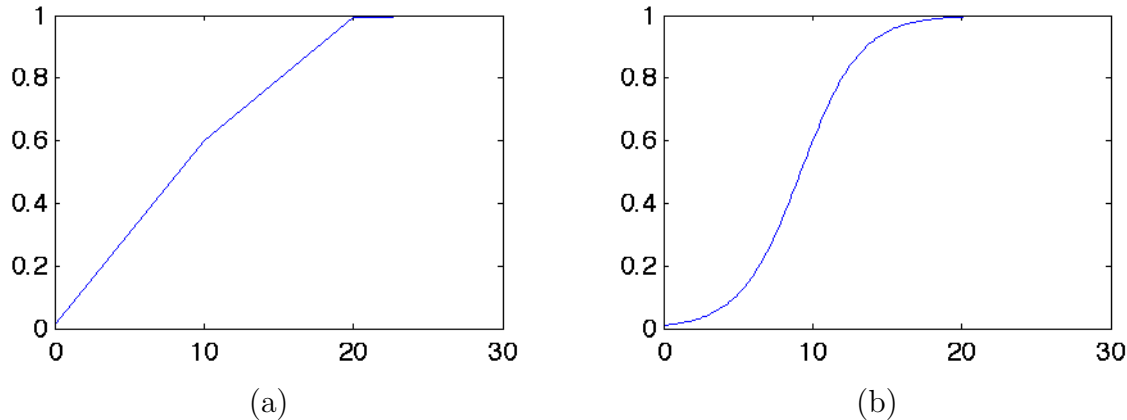


Figure 3.6. Sketch the graph of $y = 1/(1 + 99e^{-0.5t})$ on the interval $[0, 30]$.

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 \mathbf{t} , recalculate \mathbf{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.

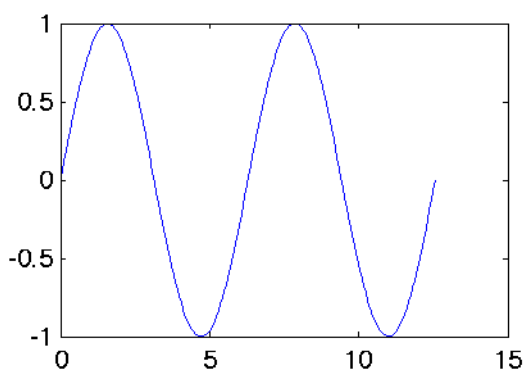
► **Example 4.** Sketch the graphs of $y = \sin x$ and $y = \cos x$ over the interval $[0, 4\pi]$.

We need to select enough x -values so that the graphs don’t exhibit the “Jaggies.”

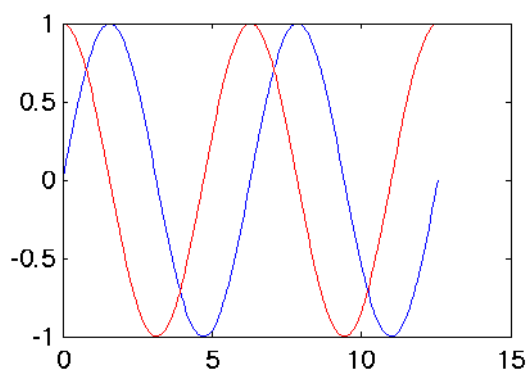
```
>> x=linspace(0,4*pi,200);
```

Next, calculate the vector \mathbf{y} and plot \mathbf{y} versus \mathbf{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 $\mathbf{y}=\cos(\mathbf{x})$ 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)')
```

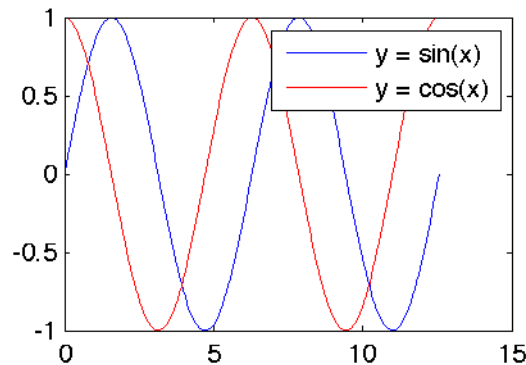


Figure 3.8. Adding a legend to help distinguish the plots.

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 prompt 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')

```

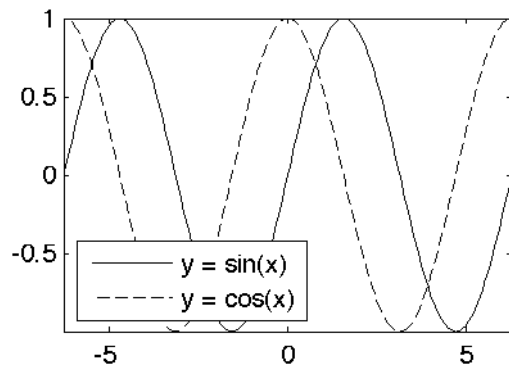
Save the script file as **twoplot.m**. While in the editor, press F5, accept a change of the current directory if prompted, then hit OK to execute the file. The resulting plot is shown in **Figure 3.9(a)**.

Some explanatory remarks are in order.

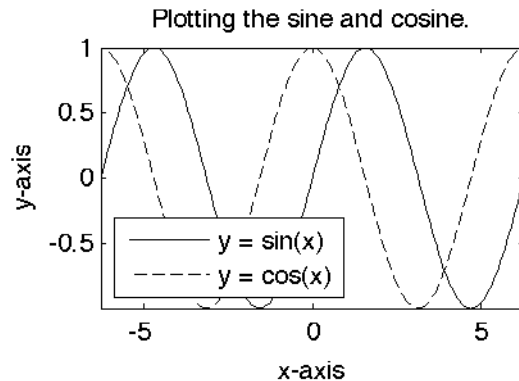
1. Note that the first four lines in the script begin with a `%`. This is a comment. Anything after the `%` is ignored by Matlab. You should get into the practice of sprinkling your code with comments.
2. The command **close all** will close **all** open figure windows, allowing a new figure window to 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 i th 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.')
```



(a)



(b)

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.

```

% twoplots.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

```

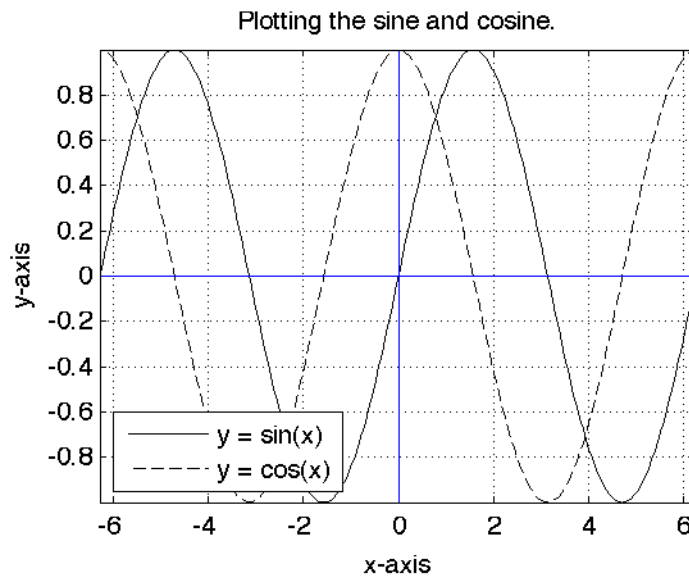


Figure 3.10. An annotated plot of sine and cosine.



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.

⁴ Copyrighted material. See: <http://msenux.redwoods.edu/IntAlgText/>

- 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. $y = \frac{1}{1+e^x}$ on $[-5, 5]$
 15. $y = x \sin x$ on $[-12\pi, 12\pi]$
 16. $y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ on $[-10, 10]$
 17. $y = \frac{1}{1+x^2}$ on $[-10, 10]$
 18. $y = (x^2 - 2x - 3)e^{-x^2}$ on $[-5, 5]$
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. $y = \frac{2+3x^2}{1+x^2}$ 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 $[-10, 10]$

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

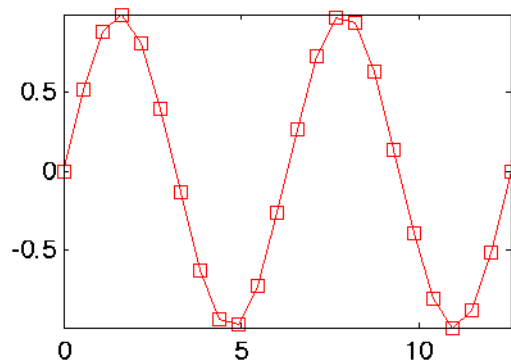
axes appropriately and provide a title. Create a legend.

- 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.

3.1 Answers

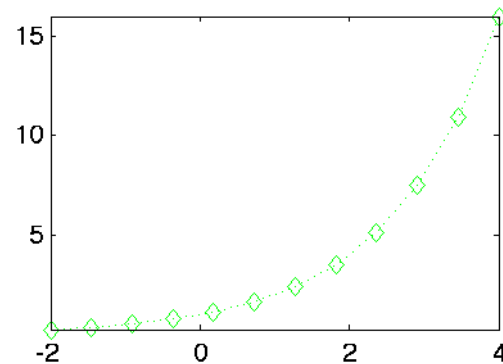
1. The following script file was used to produce the plot that follows.

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



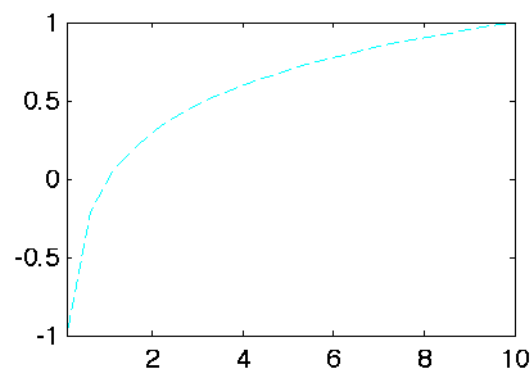
3. The following script file was used to produce the plot that follows.

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



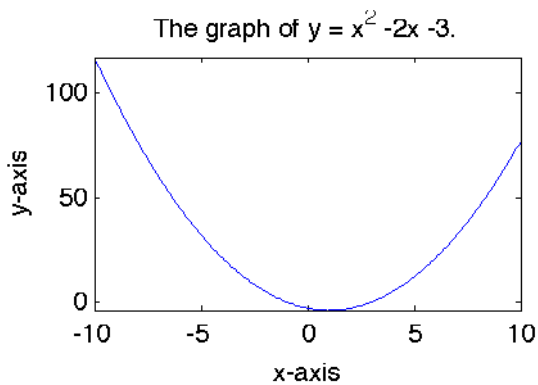
5. The following script file was used to produce the plot that follows.

```
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.

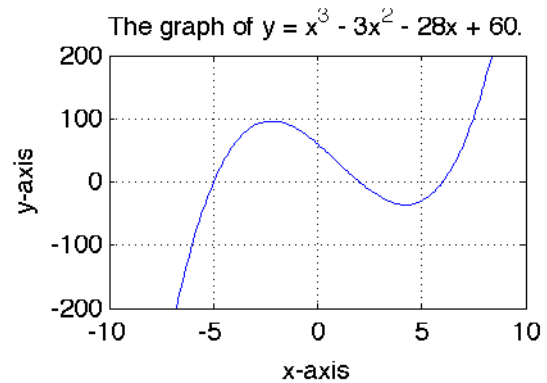
```
x=linspace(-10,10,200);
y=x.^2-2*x-3;
plot(x,y)
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('The graph of y = x^2
- 2x - 3.')
```



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
```

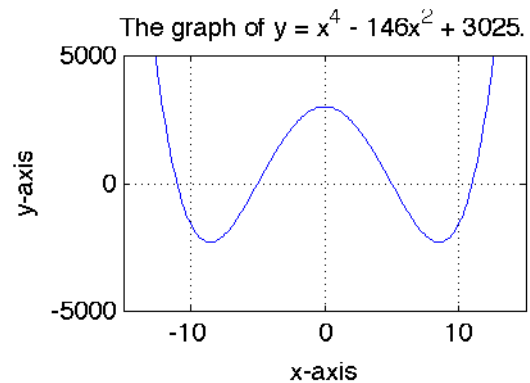
Note the use of Matlab's **axis** command and the fact that we've turned on the grid.



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
```

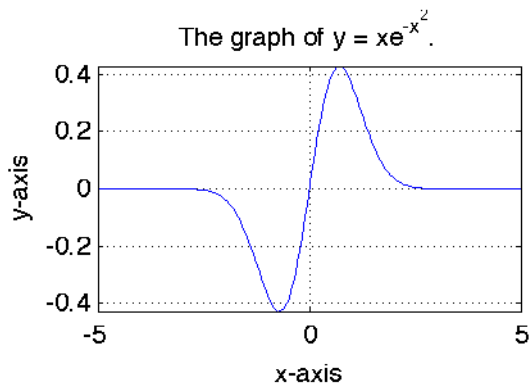
Note the use of Matlab's **axis** command and the fact that we've turned on the grid.



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
```

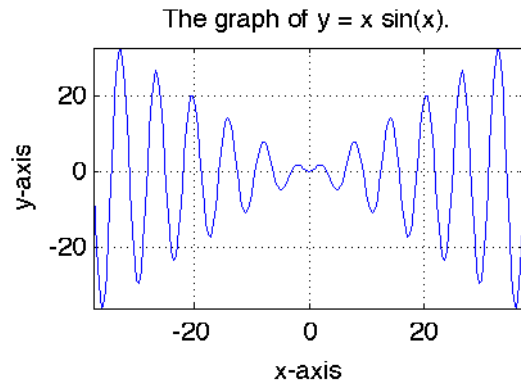
Note that we've turned on the grid.



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
```

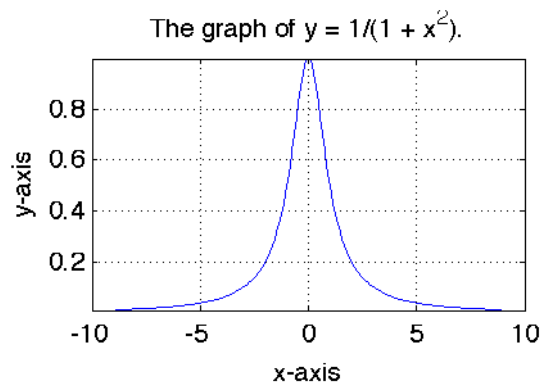
Note that we've turned on the grid.



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

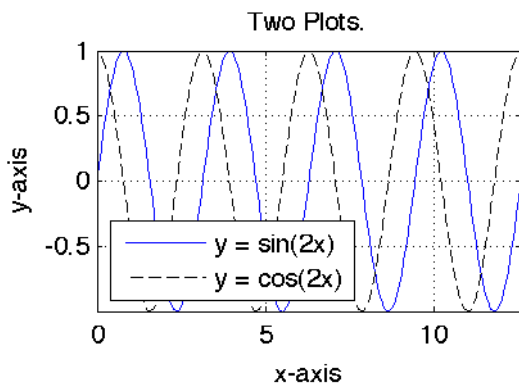
```
x=linspace(-10,10,200);
y=1./(1+x.^2);
plot(x,y)
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('The graph of
      y = 1/(1 + x^2).')
grid on
```

Note that we've turned on the grid.



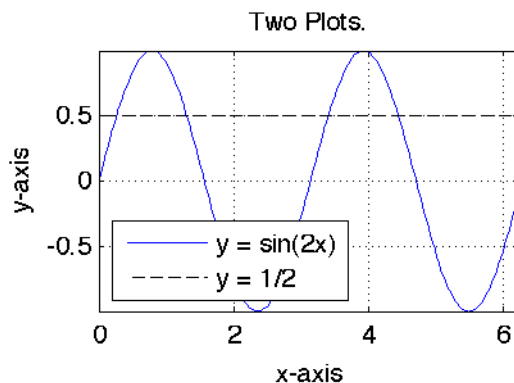
19. The following script file was used to produce the plot that follows.

```
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.

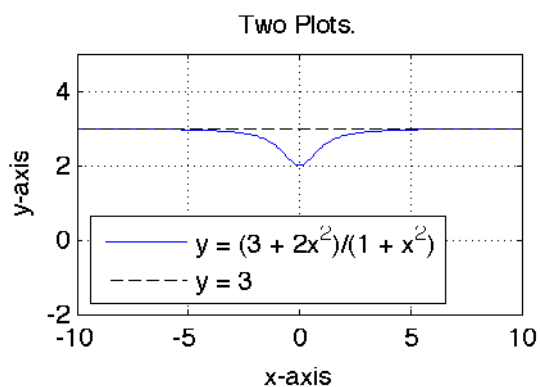
```
x=linspace(0,2*pi,200);
y1=sin(2*x);
y2=1/2*ones(size(x));
plot(x,y1,'b-',x,y2,'k--')
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('Two Plots.')
grid on
legend('y = sin(2x)',
      'y = 1/2',
      'Location',
      'SouthWest')
```



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])
```

