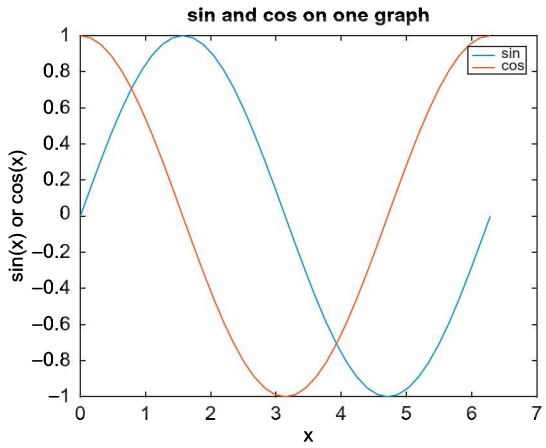

FIGURE 3.5
Plot of **sin** and **cos** in one Figure Window with a legend.

The script creates an x vector; iterating through all of the values from 0 to $2*\pi$ in steps of $2*\pi/40$ gives enough points to get a good graph. It then finds the sine of each x value, and plots these. The command **hold on** freezes this in the Figure Window so the next plot will be superimposed. Next, it finds the cosine of each x value and plots these points. The **legend** function creates a legend; the first string is paired with the first plot, and the second string with the second plot. Running this script produces the plot seen in Fig. 3.5.

Beginning with Version R2014b, when **hold on** is used, MATLAB uses a sequence of colors for the plots, rather than using the default color for each. Of course, colors can also be specified as was done in this script.

Note that instead of using **hold on**, both functions could have been plotted using one call to the **plot** function:

```
plot(x,sin(x),x,cos(x))
```

PRACTICE 3.6

Write a script that plots exp(x) and log(x) for values of x ranging from 0 to 3.5.

3.6 INTRODUCTION TO FILE INPUT/OUTPUT (LOAD AND SAVE)

In many cases, input to a script will come from a data file that has been created by another source. Also, it is useful to be able to store output in an external file

that can be manipulated and/or printed later. In this section, the simplest methods used to read from an external data file and also to write to an external data file will be demonstrated.

There are basically three different operations, or ***modes*** on files. Files can be:

- read from
- written to
- appended to

Writing to a file means writing to a file from the beginning. ***Appending to a file*** is also writing, but starting at the end of the file rather than the beginning. In other words, appending to a file means adding to what was already there.

There are many different file types, which use different filename extensions. For now, we will keep it simple and just work with .dat or .txt files when working with data or text files. There are several methods for reading from files and writing to files; we will, for now, use the **load** function to read and the **save** function to write to files. More file types and functions for manipulating them will be discussed in Chapter 9.

3.6.1 Writing Data to a File

The **save** command can be used to write data from a matrix to a data file, or to append to a data file. The format is:

```
save filename matrixvariablename -ascii
```

The "-ascii" qualifier is used when creating a text or data file. For example, the following creates a matrix and then saves the values from the matrix variable to a data file called *testfile.dat*:

```
>> mymat = rand(2,3)
mymat =
     0.4565   0.8214   0.6154
     0.0185   0.4447   0.7919

>> save testfile.dat mymat -ascii
```

This creates a file called "testfile.dat" that stores the numbers:

```
0.4565   0.8214   0.6154
0.0185   0.4447   0.7919
```

The **type** command can be used to display the contents of the file; note that scientific notation is used:

```
>> type testfile.dat

  4.5646767e-001   8.2140716e-001   6.1543235e-001
  1.8503643e-002   4.4470336e-001   7.9193704e-001
```

Note that if the file already exists, the **save** command will overwrite the file; **save** always writes from the beginning of a file.

3.6.2 Appending Data to a Data File

Once a text file is created, data can be appended to it. The format is the same as the preceding, with the addition of the qualifier "-append." For example, the following creates a new random matrix and appends it to the file that was just created:

```
>> mat2 = rand(3,3)
mymat =
    0.9218    0.4057    0.4103
    0.7382    0.9355    0.8936
    0.1763    0.9169    0.0579
>> save testfile.dat mat2 -ascii -append
```

This results in the file "testfile.dat" containing the following:

```
0.4565   0.8214   0.6154
0.0185   0.4447   0.7919
0.9218   0.4057   0.4103
0.7382   0.9355   0.8936
0.1763   0.9169   0.0579
```

Note

Although technically any size matrix could be appended to this data file, to be able to read it back into a matrix later there would have to be the same number of values on every row (or, in other words, the same number of columns).

PRACTICE 3.7

Prompt the user for the number of rows and columns of a matrix, create a matrix with that many rows and columns of random integers, and write it to a file.

3.6.3 Reading from a File

Reading from a file is accomplished using **load**. Once a file has been created (as in the preceding), it can be read into a matrix variable. If the file is a data file, the **load** command will read from the file "filename.ext" (e.g. the extension might be .dat) and create a matrix with the same name as the file. For example, if the data file "testfile.dat" had been created as shown in the previous section, this would read from it, and store the result in a matrix variable called *testfile*:

```
>> clear
>> load testfile.dat
>> who
Your variables are:
testfile
>> testfile
testfile =
```

```
0.4565   0.8214   0.6154
0.0185   0.4447   0.7919
0.9218   0.4057   0.4103
0.7382   0.9355   0.8936
0.1763   0.9169   0.0579
```

The **load** command works only if there are the same number of values in each line, so that the data can be stored in a matrix, and the **save** command only writes from a matrix to a file. If this is not the case, lower-level file I/O functions must be used; these will be discussed in Chapter 9.

3.6.3.1 Example: Load from a File and Plot the Data

As an example, a file called "timetemp.dat" stores two lines of data. The first line is the times of day, and the second line is the recorded temperature at each of those times. The first value of 0 for the time represents midnight. For example, the contents of the file might be:

```
0     3     6     9     12    15    18    21
55.5  52.4  52.6  55.7  75.6  77.7  70.3  66.6
```

The following script loads the data from the file into a matrix called *timetemp*. It then separates the matrix into vectors for the time and temperature, and then plots the data using black star (*) symbols.

timetempprob.m

```
% This reads time and temperature data for an afternoon
% from a file and plots the data

load timetemp.dat

% The times are in the first row, temps in the second row
time = timetemp(1,:);
temp = timetemp(2,:);

% Plot the data and label the plot
plot(time,temp,'k*')
xlabel('Time')
ylabel('Temperature')
title('Temperatures one afternoon')
```

Running the script produces the plot seen in Fig. 3.6.

Note that it is difficult to see the point at time 0 as it falls on the y-axis. The **axis** function could be used to change the axes from the defaults shown here.

To create the data file, the Editor in MATLAB can be used; it is not necessary to create a matrix and **save** it to a file. Instead, just enter the numbers in a new script file, and Save As *timetemp.dat*, making sure that the Current Folder is set.