

## 2.

# Plotting in MATLAB

---

MATLAB provides several methods for plotting the graphs of functions and more general curves. The easiest to use is what we will call EZ plotting, since it uses the command `ezplot` and its variants. While it is easy to use it lacks flexibility. We will briefly explain EZ plotting in the first section. The second method we will describe uses the commands `plot` and `plot3`. It provides more flexibility at the cost of some ease of use. It is the principal method used in this manual, so we will explain it in some detail. At the end of this chapter we will introduce MATLAB's handle graphics. The use of handle graphics gives the user complete control over the graphic, but it is not so easily used.

### EZ plotting

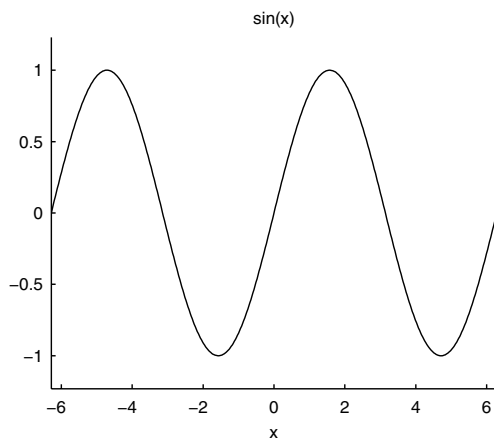
To plot the sine function in MATLAB, simply execute the command

```
>> ezplot('sin(x)')
```

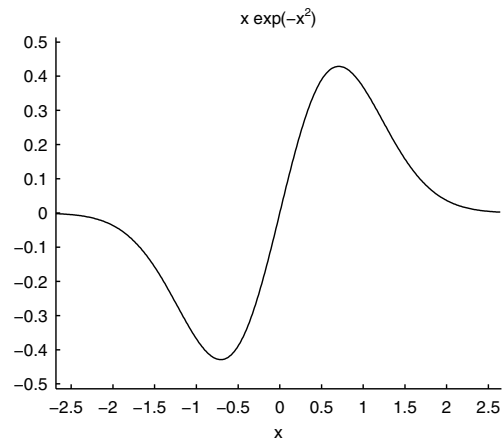
The result is shown in Figure 2.1. You will notice that `ezplot` produces the plot of  $\sin(x)$  over the interval  $[-2\pi, 2\pi]$ . Next execute

```
>> ezplot('x*exp(-x^2)')
```

This time the plot, shown in Figure 2.2, is over an interval slightly bigger than  $[-2.5, 2.5]$ .



**Figure 2.1.** Plot of  $\sin x$ .



**Figure 2.2.** Plot of  $x e^{-x^2}$ .

Notice that you did not have to specify a plot interval. The command `ezplot` uses  $[-2\pi, 2\pi]$  as the default interval over which to produce a plot. If the function is almost constant near the endpoints of

the interval  $[-2\pi, 2\pi]$ , as is  $xe^{-x^2}$ , then `ezplot` chooses a smaller interval. If the default interval does not please you, you can choose your own. For example, execute

```
>> ezplot('sin(x)', [0, 8*pi])
```

to plot the sine function over the interval  $[0, 8\pi]$ .

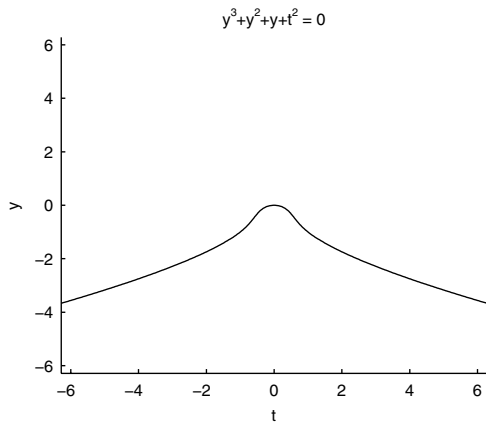
**Example 1.** The initial value problem  $y' = y + t$  with  $y(0) = 0$  has the solution  $y(t) = e^t - t - 1$ . Plot this solution over the interval  $0 \leq t \leq 2$ .

This can be done with the single command `ezplot('exp(t)-t-1', [0, 2])`.

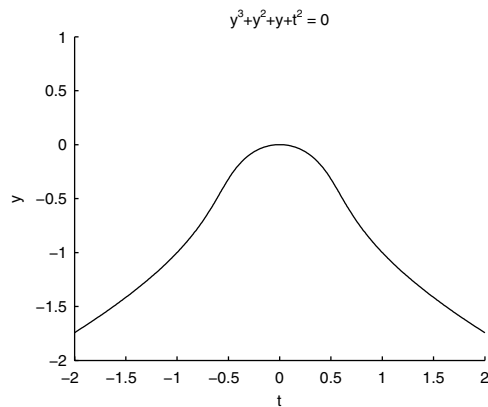
The command `ezplot` can also plot curves that are defined implicitly or parametrically. To learn about these features execute `help ezplot`. This command also illustrates the use of the command `help`. Whenever you run across a command you do not understand, use the `help` command. It will provide you with the information you need.

**Example 2.** The solution to the initial value problem  $y' = -2t/(3y^2 + 2y + 1)$  with  $y(0) = 0$  satisfies the implicit relationship  $y^3 + y^2 + y + t^2 = 0$ . Plot the solution over the interval  $-2 \leq t \leq 2$ .

If you execute the command `ezplot('y^3+y^2+y+t^2 = 0')`, you will get the plot of the solution shown in Figure 2.3. Notice that by default the solution is plotted over  $-2\pi \leq x \leq 2\pi$ . The important part of the solution would be better portrayed if the  $t$ -axis were reduced to  $-2 \leq t \leq 2$  and the  $y$ -axis were reduced to  $-2 \leq y \leq 1$ . This can be done with the command `ezplot('y^3+y^2+y+t^2', [-2, 2, -2, 1])`. The result is shown in Figure 2.4. It is very typical that our first attempt is not completely satisfactory. Do not be afraid to redo a plot to make it look better.



**Figure 2.3.** The solution in Example 2.



**Figure 2.4.** The solution in a smaller plot area.

The vector  $[-2, 2, -2, 1]$  that appears in the `ezplot` command means that the plot area will be limited by  $-2 \leq t \leq 2$  and  $-2 \leq y \leq 1$ . This vector is referred to as the *axis*. You can discover

the axis of the current plot by executing `axis` on the command line. Execute `help axis` to learn more about this versatile command. For example, the same reduction of plot area can be accomplished with the command `axis([-2, 2, -2, 1])` after the plot in Figure 2.3 was done.

If you own the Symbolic Toolbox, an add-on product to MATLAB, the command `ezplot` provides an excellent way to get a quick plot of the solution of an initial value problem.

**Example 3.** Graph the solution of  $y' - y = e^{-t} \cos 3t$ , with  $y(0) = 1$ .

Enter the command

```
>> dsolve('Dy - y = exp(-t)*cos(3*t)', 'y(0) = 1')
ans =
-2/13*exp(-t)*cos(3*t)+3/13*sin(3*t)*exp(-t)+15/13*exp(t)
```

Then the command

```
>> ezplot(ans)
```

will provide a plot of the solution. We discuss the Symbolic Toolbox in Chapter 10.

## Matrices and Vectors in MATLAB

To use the `plot` command effectively it is necessary to know a little about how MATLAB works. A powerful feature of MATLAB is that every numerical quantity is considered to be a complex matrix!<sup>1</sup> For those of you who do not already know, a *matrix* is a rectangular array of numbers. For example,

$$\mathbf{A} = \begin{bmatrix} 1 & \pi & \sqrt{-1} \\ \sqrt{2} & 4 & 0 \end{bmatrix}$$

is a matrix with 2 rows and 3 columns.

If you want to enter the matrix  $A$  into MATLAB proceed as follows:

```
>> A = [1,pi,sqrt(-1);sqrt(2),4,0]
A =
    1.0000          3.1416          0 + 1.0000i
    1.4142          4.0000          0
```

Note that commas are used to separate the individual elements in a row, and semicolons are used to separate the rows of the matrix. You can also use *spaces* to separate (delimit) the entries in a row. Thus, the command `A = [1 pi sqrt(-1);sqrt(2) 4 0]` can be used to enter  $A$  into MATLAB.

The *size* of a matrix is the number of rows and columns. For example,

```
>> size(A)
ans =
     2     3
```

---

<sup>1</sup> A matrix whose entries are complex numbers.

verifies that  $A$  has 2 rows and 3 columns. Two matrices are said to have the same size if they have the same number of rows and the same number of columns.

Even single numbers in MATLAB are matrices. For example,

```
>> a = 5;
>> size(a)
ans =
     1     1
```

shows that MATLAB thinks that 5, or any other complex number, is a matrix with one row and one column.

A *vector* is a list of numbers.<sup>2</sup> It can be a vertical list, in which case it is called a *column vector*, or it can be a horizontal list, in which case it is called a *row vector*. Vectors are special cases of matrices, so you can enter the row vector  $v = [1, -5, \pi, \sqrt{-1}]$  into MATLAB using the command

```
v = [1,-5,pi,sqrt(-1)]
```

or  $v = [1 \ -5 \ \pi \ \text{sqrt}(-1)]$ . On the other hand, we can define a column vector with the command  $u = [1;2;3;4]$ . It is important to remember that MATLAB distinguishes between row and column vectors.

In MATLAB, the *length* of a vector is the number of elements in the list. For example, the MATLAB length of each of the vectors  $u$  and  $v$  defined in the previous paragraph is 4. The MATLAB command `length` will disclose the length of any vector. Try `length(u)` and `length(v)`. Notice that  $u$  and  $v$  have the same length, but not the same size, since  $u$  is a column vector and  $v$  is a row vector.

This notion of length is not to be confused with the *geometric length* of a vector, defined to be the square root of the sum of the squares of the absolute values of the entries. MATLAB's command for finding the geometric length of a vector is `norm`. For example,

```
>> norm(v)
ans =
     6.0720
```

### Addition, Subtraction, and Multiplication by Scalars

If  $A$  and  $B$  are matrices of the same size, then they can be added together. For example,

---

<sup>2</sup> The word *vector* is one of the most over used terms in mathematics and its applications. To a physicist or a geometer, a vector is a directed line segment. To an algebraist or to many engineers, a vector is a list of numbers. To users of more advanced parts of linear algebra, a vector is an element of a vector space. In this latter, most general case, a vector could be any of the above examples, a polynomial, a more general function, or an example of, quite literally, any class of mathematical objects which can be added together and scaled by multiplication.

All too often the meaning in any particular situation is not explained. The result is very confusing to the student. When the word vector appears, a student should make a concerted effort to discover the meaning that is used in the current setting.

When using MATLAB, the situation is clear. A vector is a list of numbers, which may be complex.

```

>> A = [1 2;3 4], B = [5,6;7,8], C = A + B
A =
     1     2
     3     4
B =
     5     6
     7     8
C =
     6     8
    10    12

```

You will notice that each element of the matrix  $C$  is sum of the corresponding elements in the matrices  $A$  and  $B$ . The same is true for the difference of two matrices. Try  $C-A$ , and see what you get (you should get  $B$ ).

You can multiply or divide any matrix by a scalar.

```

>> v = ones(1,5); w = 4*v, z = v/2
w =
     4     4     4     4     4
z =
    0.5000    0.5000    0.5000    0.5000    0.5000

```

While multiplication and division by scalars are standard parts of matrix algebra, the addition of a scalar to, or the subtraction of a scalar from a matrix are not standard algebra. However, they are allowed in MATLAB. Try

```

>> m = v - 3, a = v + 4
m =
    -2    -2    -2    -2    -2
a =
     5     5     5     5     5

```

MATLAB's *transpose* operator (a single apostrophe) changes a column vector into a row vector (and vice-versa).

```

>> u = [1;2], v = u'
u =
     1
     2
v =
     1     2

```

Actually,  $\cdot'$  is MATLAB's transpose operator and  $'$  is MATLAB's *conjugate transpose* operator. Enter  $A = [1+i, -2; 3i, 2-i]$ , then type  $A'$  and view the results. Note that rows of  $A$  have become columns, but each entry has been replaced with its complex conjugate. Type  $A \cdot'$  to appreciate the difference. If each entry of a matrix is a real number, then it doesn't matter whether you use  $'$  or  $\cdot'$ .

## Array Operations and Array Smart Functions

MATLAB has built-in, element-by-element, operations for other mathematical operations on matrices. For example, should you need to multiply two vectors on an element-by-element basis, use MATLAB's `.*` operator.

```
>> v = [1,2,3,4], w = [5,6,7,8], u = v.*w
v =
    1  2  3  4
w =
    5  6  7  8
u =
    5 12 21 32
```

If you look closely, you will see that `u` is a vector of the same size as `v` and `w`, and that each element in `u` is the product of the corresponding elements in `v` and `w`. This operation is called *array multiplication*, and MATLAB's symbol for it is `.*`, not `*`. MATLAB uses `*` for matrix multiplication, which is quite different from array multiplication. Try entering `v*w` and see what happens. We will explain matrix multiplication in Chapter 11.

There are other array operations. All of them act element-by-element. Try `v./w` and `w./v`. This is *array right division*. Then try `v.\w` and `w.\v`, and compare the results. This is called *array left division*. There is one other array operation — *array exponentiation*. This is also an element-by-element operation. The operation `A.^2` results in every element of the matrix `A` being raised to the second power. For example, if `A = [1,2;3,4]`, the command

```
>> A = [1,2;3,4], B = A.^2
A =
    1     2
    3     4
B =
    1     4
    9    16
```

raises each entry in `A` to the second power. The command `A^2` is equivalent to `A*A`, which gives an entirely different result. Try it and see. For all array operations it is required that the matrices have exactly the same size. You might try `[1,2;3,4].*[1;1]` to see what happens when they are not.

The built-in MATLAB functions, which we discussed briefly in Chapter 1, are all designed to be *array smart*. This means that if you apply them to a matrix, the result will be the matrix obtained by applying the function to each individual element. For example:

```
>> theta = [0,pi/2,pi,3*pi/2,2*pi], y = cos(theta)
theta =
    0    1.5708    3.1416    4.7124    6.2832
y =
    1.0000    0.0000   -1.0000    0.0000    1.0000
```

This is an extremely important feature of MATLAB, as you will discover in the next section.

## Plotting in MATLAB Using the plot Command

None of these array operations would be important if it were not so easy to create and use vectors and matrices in MATLAB. Here is a typical situation. Suppose we want to define a vector that contains a large number of equally spaced points in an interval  $[a, b]$ . MATLAB's `start:increment:finish` construct allows you to generate equally spaced points with ease. For example, the command

```
>> t = 0:0.2:1
t =
    0    0.2000    0.4000    0.6000    0.8000    1.0000
```

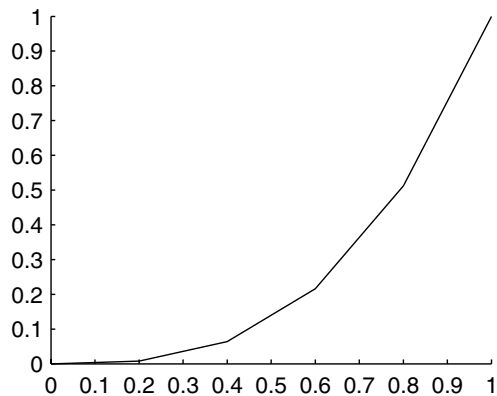
generates numbers from 0 to 1 in increments of 0.2.<sup>3</sup> The command `y = t.^3` will produce a vector `y` with 6 entries, each the cube of the corresponding entry in the vector `t`.

```
>> y = t.^3
y =
    0    0.0080    0.0640    0.2160    0.5120    1.0000
```

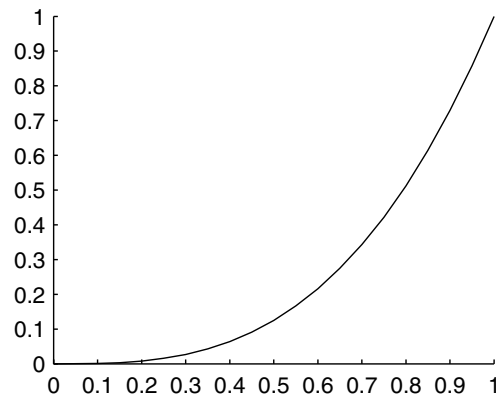
We can get a rudimentary plot of the function  $y = t^3$  by plotting the entries of `y` versus the entries of `t`. MATLAB will do this for us. The command

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

will produce a plot of `y` versus `t` in the current figure window. If no figure window exists, then the command `plot` will create one for you. MATLAB plots the 6 ordered pairs  $(t, y)$  generated by the vectors `t` and `y`, connecting consecutive ordered pairs with line segments, to produce an plot similar to the that shown in Figure 2.5.



**Figure 2.5.** A simple plot



**Figure 2.6.** A simple plot with refined data.

---

<sup>3</sup> If you omit the increment, as in `t = 0:10`, MATLAB automatically increments by 1. For example, try `q = 0:10`.

Notice that the plot in Figure 2.5 is kinky. This is because we plotted too few points before MATLAB connected the dots with straight lines. If we use enough points we get a smooth looking curve. For example, the commands

```
>> t = 0:0.05:1; y = t.^3; plot(t,y), shg
```

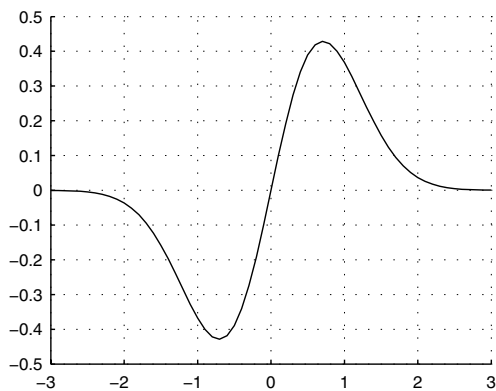
produced the graph in Figure 2.6. This time there are 21 points, and the curve appears smooth. Notice that multiple commands can be entered on a single command line if they are separated with commas and/or semicolons. The command `shg` stands for “show the graph,” and it brings the current figure window<sup>4</sup> to the front. It is a good idea to add `shg` to plot commands on the command window.

**Example 4.** Use the `plot` command to graph  $f(x) = xe^{-x^2}$  over the interval  $[-2, 2]$ .

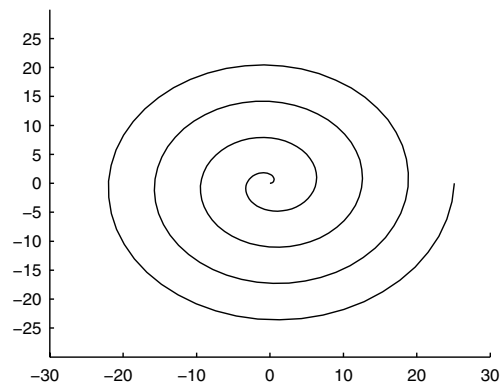
To accomplish this task, we need two MATLAB vectors. First, we need a vector `x` containing a large number of values between  $-2$  and  $2$ . We can do this with the command `x = -2:0.1:2`. Next we need a vector `y` containing the values of  $f(x) = xe^{-x^2}$  at the points in `x`. This can be accomplished using the array operations. The operation `.^` works element by element, so the vector `x.^2` contains the squares of the values in `x`. Since MATLAB functions are array smart, `exp(-x.^2)` contains the values of  $e^{-x^2}$  for each of the entries of `x`. Finally, since `*` is an array operation, `x.*exp(-x.^2)` contains the values of  $f(x) = xe^{-x^2}$  for the entries in `x`. Thus the commands

```
>> x = -2:0.1:2;
>> y = x.*exp(-x.^2);
>> plot(x,y), shg
```

will produce the desired graph. Executing the command `grid` produces Figure 2.7.



**Figure 2.7.** The graph for Example 4.



**Figure 2.8.** The parametric curve in Example 5.

<sup>4</sup> If several figure windows are open, the last one visited is the “current figure window.”

**Parametric plots.** Notice that in Example 4 we used the command `plot(x,y)`, where `x` and `y` were vectors of the same size. This command all by itself cares not where the two vectors came from. For any two vectors of the same size, the command will plot the  $(x, y)$  pairs and connect them with line segments. We can utilize this to produce parametric plots.

**Example 5.** Plot the parametric curve defined by  $t \rightarrow (t \cos t, t \sin t)$  for  $0 \leq t \leq 8\pi$ .

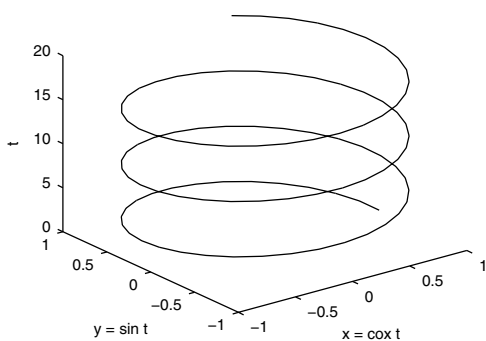
We start with the command `t = linspace(0,8*pi,200)`, which produces 200 equally spaced points<sup>5</sup> between 0 and  $8\pi$ . Then `x = t.*cos(t)`, and `y = t.*sin(t)` produce the corresponding values of the components of the desired curve. Finally, `plot(x,y)`, `shg` produces the plot. To summarize, we use the commands

```
>> t = linspace(0,8*pi,200);
>> x = t.*cos(t);
>> y = t.*sin(t);
>> plot(x,y), shg
```

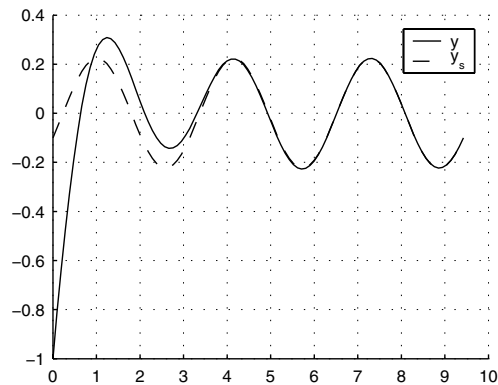
**Curves in Three Dimensions.** Three dimensional plots require the use of `plot3` instead of `plot`, but otherwise the method is unchanged. The commands

```
>> t = linspace(0,20);
>> x = cos(t); y = sin(t); z = t;
>> plot3(x,y,z), shg
>> xlabel('x = cos t'); ylabel('y = sin t'); zlabel('t')
```

produce the helix in Figure 2.9. By default, the command `linspace(a,b)` produces a vector of 100 evenly spaced point between  $a$  and  $b$ .



**Figure 2.9.** The spiral curve with  $x = \cos t$ ,  $y = \sin t$ , and  $z = t$ .



**Figure 2.10.** Plots of  $y$  and  $y_s$ .

<sup>5</sup> Type `help linspace` for more information on using this command

**Colors, markers, and line styles.** Execute `help plot` at the MATLAB prompt and read the resulting help file. Pay particular attention to the various line styles, markers, and colors that can be used with MATLAB's `plot` command. You can produce strikingly different plots by varying the choice of these three attributes. Try the command `plot(t,y,'rx:')`, `shg`, and examine its affect on your plot. This plot command has the third argument `'rx:'`. The three symbols between the single quotes select, in order, a color, a marker, and a line style. Experiment with other combinations, such as `plot(t,y,'s')`, `plot(t,y,'md')`, and `plot(t,y,'k--')`. Use the `shg` command to view the results of each command.

**Multiple graphs in a figure.** There are several ways of doing this in MATLAB.

**Example 6.** *The initial value problem*

$$y'' + 2y' + 2y = \cos 2t, \quad \text{with } y(0) = -1 \quad \text{and} \quad y'(0) = 2$$

has the solution  $y(t) = y_t(t) + y_s(t)$ , where

$$y_t(t) = e^{-t} \left( \frac{7}{10} \sin t - \frac{9}{10} \cos t \right), \quad \text{and} \quad y_s(t) = \frac{1}{5} \sin 2t - \frac{1}{10} \cos 2t$$

are the transient response and the steady-state solution, respectively. Plot the graphs of  $y$  and  $y_s$  over the interval  $[0, 3\pi]$  on the same figure. Use a solid line for  $y$  and a dashed line for  $y_s$ . Use a legend to label the graphs.

We can use `t = linspace(0,3*pi)` to get 100 equally spaced  $t$ -values between 0 and  $3\pi$ . The commands

```
>> y_t = exp(-t).*((7/10)*sin(t) -(9/10)*cos(t));
>> y_s = (1/5)*sin(2*t) - (1/10)*cos(2*t);
>> y = y_t + y_s;
```

compute the steady-state solution, the transient response, and the total response. A naive first try at plotting  $y$  and  $y_s$  together might be

```
>> plot(t,y)
>> plot(t,y_s,'--'), shg
```

However, MATLAB erases the first plot when the second plot command is executed. The commands

```
>> plot(t,y,t,y_s,'--')
>> grid on
>> legend('y','y_s')
```

will produce an image similar to that in Figure 2.10, with the graphs of both  $y$  and  $y_s$  on the same figure.

Notice that the parameters of the plot command come in groups. The first group consists of `t` and `y`, which is the data for the first curve. The second group has three entries, `t`, `y_s`, and `'--'`. This is the data for the the second curve plus a designation of a line style. The command `grid on` does just what it says — it adds a grid to the figure. Type `grid off` if you want to remove the grid. The `legend` command

produces the legend. It is only necessary to list the names in the order the graphs were produced. Notice that although entered as `y_s`, the label in the legend is subscripted.

A second solution to the problem of adding graphs to a figure involves the commands `hold on` and `hold off`. The command `hold on` tells MATLAB to add subsequent plots to the existing figure, without erasing what is already there. The command `hold off` tells MATLAB to return to the standard procedure of erasing everything before the next plot. This means that `hold on` is in effect until a `hold off` command is executed. Thus, we could have used the commands

```
>> plot(t,y)
>> hold on
>> plot(t,y_s,'--'), shg
>> hold off
```

to produce the plot in Figure 2.10.

A third way to plot two curves is to put the two sets of  $y$ -data into a matrix. The single command

```
>> plot(t,[y;y_s])
```

will cause the two curves to be plotted. The command `[y;y_s]` puts the two row vectors  $y$  and  $y_s$  into a matrix with two rows and as many columns as  $t$ . If  $A$  is a matrix with as many columns as  $t$ , then `plot(t,A)` will graph each row of  $A$  against  $t$ . A different color is automatically chosen for each curve.

## Editing Graphics

MATLAB has extensive figure editing features. To use these make sure that the Figure Toolbar is visible by selecting **View**→**Figure Toolbar**.<sup>6</sup>

To change the appearance of a curve, click on the selection tool (the arrow pointing up and to the left in the Toolbar). Then click on the curve to select it. Now right-click<sup>7</sup> on the curve to bring up a context menu. There are several choices, but the ones we are most interested in allow us to change the line-width, line-style, and color of the curve. The use of these is amply clear once you try them once. If you select **Properties**, you are provided with a Property Editor window that gives complete control over the line properties, including adding markers to the data points and all of the available marker properties.

The Property Editor can also be accessed through the **Edit** menu. To edit the properties of a curve, first make that curve the *current object* by clicking on it. Then select **Edit**→**Current Object Properties** .... There are also Property Editors for the Figure (the area around the graph) and the Axes (the axis lines and labels, together with the background of the graph). In each case selection opens a new window which gives you complete control over the appearance of the object chosen.

For more information about editing your graphs, select **Help**→**Formatting Graphs** in any figure window.

---

<sup>6</sup> The notation **View**→**Figure Toolbar** means you should choose **Figure Toolbar** from the **View** menu located on the current figure window.

<sup>7</sup> Here we are assuming you have a mouse with two or more buttons. If you use a Macintosh and have only one button, use a control-click instead of a right-click.

## Saving, Printing, and Exporting Your Plot

After learning how to plot graphs, you will want to print them. Simply use the print command in the File menu, or the print icon in the toolbar. **Edit**→**Print Setup** allows you to choose a printer and its properties and to choose between landscape or portrait output. **Edit**→**Page Setup** allows you to change many aspects of the printout.

To save a MATLAB figure to the clipboard, use the **Edit**→**Copy Figure** command. You will then be able to paste the figure into another document.

If you need to save a copy of a MATLAB figure to a graphics file, use the **File**→**Export** command. You will be given the opportunity to choose from a large variety of graphics formats. The choices made using **Edit**→**Page Setup** affect the exported file, so some flexibility is available.

It is possible to print what appears in the current figure window<sup>8</sup> to the default printer by simply entering `print` at the MATLAB prompt. In fact the `print` command provides many possibilities. Execute `help print` to explore them. For example, it is possible to export a figure to a graphics file using the `print` command at the command line. The command

```
>> print -deps junk.eps
```

will save the current figure as the encapsulated postscript file `junk.eps` in the current directory.

## Script M-files

You will have noticed that producing a finished graphic in MATLAB requires the use of several commands entered at the command line. A finished graphic often requires several passes through these commands in order to get everything just right. Fortunately there is a way to get around the need to repeatedly enter a large list of commands as you improve on a graphic. It is possible to enter these commands into a text file, and execute all of them with one command.

Such files are called *M-files*. M-files come in two types, each with its own features. The type to use in building a complicated graphic is a *script M-file*, and we will describe them in this chapter. In addition there are *function M-files*, which can be used, for example, to extend MATLAB's library of functions. We will discuss function M-files in Chapter 4.

Let's start with a complicated graphing problem.

**Example 7.** *In one figure, plot the solutions to the differential equation  $y' = y + t$  with initial conditions  $y(0) = -2, -1, 0, 1, 2$  over the interval  $0 \leq t \leq 2$ .*

The general solution to the differential equation is  $y(t) = Ce^t - t - 1$ . The initial condition is  $y(0) = C - 1$ , so the constant satisfies  $C = y(0) + 1$ . The graphs can be drawn using the techniques we have already discussed, but it is easy to make mistakes in executing all of the commands needed. Instead we will create a script M-file. To see how this is done, choose the menu item **File**→**New**→**M-file**. The built-in MATLAB editor<sup>9</sup> will open at a blank page. You can also call up the editor by executing the

---

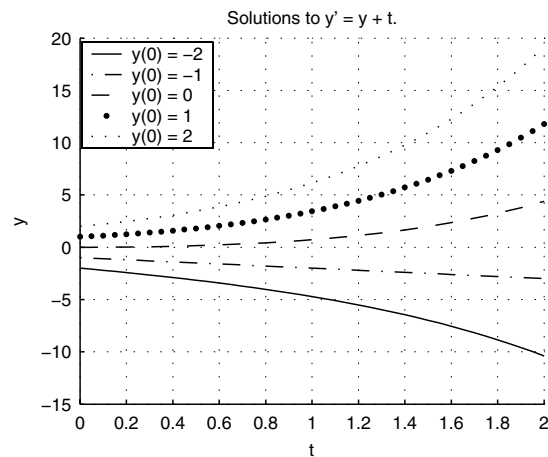
<sup>8</sup> MATLAB's `figure` command allows you to create multiple figure windows. If you have several figure windows open, click any figure window with your mouse to make it the current figure window.

<sup>9</sup> Starting with version 5.2, MATLAB has built-in editor on every platform. Of course, it is not

command edit. We will let MATLAB perform the simple arithmetic to compute the constant in addition to plotting the solutions. Enter the following list of commands into the blank editor page:

```
t = 0:0.05:2;
C = -2+1; plot(t,C*exp(t) - t - 1,'-')
hold on
C = -1+1; plot(t,C*exp(t) - t - 1,'-.')
C = 0+1; plot(t,C*exp(t) - t - 1,'--')
C = 1+1; plot(t,C*exp(t) - t - 1,'.')
C = 2+1; plot(t,C*exp(t) - t - 1,':')
grid on
xlabel('t')
ylabel('y')
title('Solutions to y' = y + t.')
legend('y(0) = -2','y(0) = -1','y(0) = 0','y(0) = 1','y(0) = 2')
shg, hold off
```

Finally, save the file with a meaningful name such as `ch2examp7.m`. Now whenever you execute the command `ch2examp7` at the command line, all of these commands are executed and the figure appears. The figure is shown in Figure 2.11.



**Figure 2.11.** The family of solutions to a differential equation.

The commands `xlabel`, `ylabel`, and `title` at the end of the file `ch2examp7.m` are easily understood. You can get more information about them using the `help` command.

---

necessary to use the built-in editor. Any text editor will do. It is even possible to use a word processor, but if you do it is absolutely essential that you save the file as a text file.

## Text strings in MATLAB

Notice that the labeling commands in `ch2examp7.m` accept strings of text as inputs. In MATLAB a string is any list of symbols entered between single quotes. This raises a question. How do we enter a string that contains a single quote, such as the title of the graph produced by `ch2examp7.m`? The answer is that we simply replace the single quote with a double quote, as in `'Solutions to y'' = y + t.'`.

We have now seen two different classes of data used by MATLAB. The first consists of numeric quantities like matrices and numbers. Since these are computed and stored to double the standard precision used in computers, the class is called *double*. The second class consists of strings of characters, and is called *char*. We will need to know how to change a number into a string. This is done with the command `num2str`. The use of this mnemonic command is illustrated by

```
>> x = 5.324, xstring = num2str(x)
x =
    5.3240
xstring =
    5.324
```

So `x` is a number, and `xstring` is the same number transformed into a string. Notice that the only way to differentiate them on the command window is by the different indentation that MATLAB gives them. To get more information we can use the command

```
>> whos
      Name          Size          Bytes  Class
      x             1x1             8    double array
      xstring       1x5             10    char array
```

```
Grand total is 6 elements using 18 bytes
```

This clearly illustrates the different classes of data. It is also possible to change a string into a number using `str2num`, but we will have less use for that.

Finally, we need to know how to concatenate strings to make bigger strings. An example is

```
>> string1 = 'Hello'; string2 = 'there';
>> string3 = [string1, ' ', string2, '.']
string3 =
Hello there.
```

The concatenated string, `string3`, is formed by placing the four strings `string1`, `string2`, `' '`, and `'.'` between square brackets, separated by commas. The string `' '` provides the space between `string1` and `string2`, while `'.'` provides the period at the end.

## A Little Programming

In the script M-file `ch2examp7.m` that we created for Example 7 we had to insert a line for each curve. We can replace that file with this one, which we will call `ch2examp7_2.m`.

```
t = 0:0.05:2;
Y = [];
for k = -2:2
    C = k+1; Y = [Y;C*exp(t)-t-1];
end
plot(t,Y)
grid on
xlabel('t'); ylabel('y')
title('Solutions to y' = y + t.')
shg
```

The command `Y = []` introduces an empty matrix. Then at each of the five steps in the `for` loop, corresponding to  $k = -2, -1, 0, 1, 2$ , we first compute the constant  $C = k+1$ , and then the vector  $C \cdot \exp(t) - t - 1$ , which is added as a new row to the matrix `Y`. At the end `Y` has a row for each solution and the `plot` command plots each of the five rows versus `t`, choosing a distinctive color for each. The use of such a simple `for` loop will simplify M-files whenever repetitive computations are required.

## Other Issues with M-files

**Organizing your files.** Once you start using M-files, they tend to accumulate. A little organization can help keep some order in what could easily become chaos. If you haven't done so already (see Chapter 1, Exercise 1), use the standard procedure on your operating system to create a folder or directory called `mywork` (or invent a name of your own) in which to store your files. Make this directory the current directory by using the command `cd`<sup>10</sup> in the MATLAB command window or by clicking the ellipsis (...) button next to the Current Directory edit box on the toolbar of the command window and browsing to the new directory. You can check your success by looking at the contents of Current Directory edit box, or by executing `pwd` in the command window. It is important to understand that MATLAB first searches the current directory for script files. Hence, if you save your script file in a particular folder, make that folder the "current directory" before attempting to execute the file or MATLAB will be unable to find it. Alternatively, you can also put the folder `mywork` on MATLAB's path, but we will have more to say about this later.

Notice that we named the M-file used in Example 7 `ch2examp7.m`. This mnemonic enables us to identify this file in the future as the file used in Example 7 in Chapter 2. You will find it useful to invent a file naming system of your own.

**The MATLAB editor.** While any editor can be used to create M-files, there are definite advantages to using MATLAB's built-in editor. We will mention two.

- After creating your file, it can be saved and executed by selecting the menu item **Debug**→**Run** in the editor. You can use the accelerator key F5 to do the same thing after any changes you make.

---

<sup>10</sup> Type `help cd` to learn more about changing the "current directory."

- It is a rare thing to create an M-file that has no errors the first time through. The beauty of the M-file approach is that these errors can be eliminated one by one. Should your file contain errors, you should hear a “beep” or “click” when you press the F5 button in MATLAB’s editor. An error message will be reported in the MATLAB command window, often as a link. Clicking this link will take you directly to the line in the MATLAB editor where the error occurs. This is an extremely useful tool for debugging your function M-files.

### Handle graphics.

We have seen how the Property Editors enable great flexibility in editing figures. If we want the same flexibility when constructing figures from the command line or using M-files we need to use *handle graphics*.

**Example 8.** *The output of a forced undamped harmonic oscillator is given by*

$$x(t) = 2 \sin \frac{t}{2} \sin \frac{23t}{2}.$$

*Plot the solution over the interval  $[-2\pi, 2\pi]$ . In addition plot the envelope  $\pm 2 \sin(t/2)$  with a line width of 2 pixels, and in a distinctive color.*

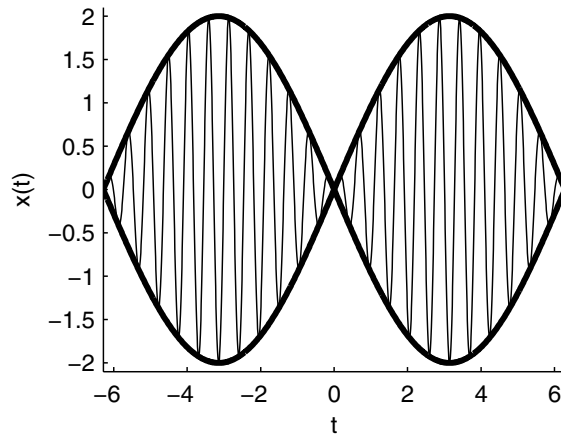
The only difficult part of this is increasing the line width. The normal line width is 0.4 pixels. Effecting the change using the Property Editor is easy, but we want to do this in an M-file, so that we can reproduce it easily. One way to do this is with these instructions.

```
t = linspace(-2*pi,2*pi,1000);
x = 2*sin(t/2).*sin(23*t/2);
env = 2*sin(t/2);
plot(t,x)
hold on
h = plot(t,[env;-env]);
set(h,'linewidth',2,'color','c');
hold off
axis([-2*pi,2*pi,-2.1,2.1])
xlabel('t')
ylabel('x(t)')
shg
```

We needed the very high resolution in `linspace` because the graph of  $x$  is highly oscillatory, and anything less was not sufficient. We inserted the `axis` command to better place the graphs in the figure.

However, the interesting commands are those that plot the envelope. Notice that the command `h = plot(t,[env;-env])` outputs the quantity `h`. This is a vector of numerical handles to the two curves of the envelope. Execute `h` to see what the vector is. The next command, using `set`, changes the line width to 2 pixels and the color to cyan for both of the curves with handles in `h`. This is an example of the use of handle graphics. A black and white version of the result is shown in Figure 2.12.

Notice how the `set` command allows you to change individual properties of a curve. If you execute `set(h(1))` you will see a full list of the properties, together with possible settings. The command



**Figure 2.12.** The phenomenon of beats.

`get(h(1))` will display a list of the settings for the curve with handle `h(1)`. Of course, you can replace `h(1)` with `h(2)` to see the results for the second curve. As is illustrated in Example 8, the `set` command can be used to change individual properties using 'property', 'property value' pairs.

Curves are objects of the class *line*. There are a large number of object classes. An *axes* is the area on which a curve is plotted, including the axis lines and the associated markings. If you execute `gca`, you will see the handle of the current axes. Execute `set(gca)` to see the list of all of the properties of an axes. Any of these can be changed using the `set` command, just as we did for graphs. A *figure* is a figure window. You can find the handle of the current figure with the command `gcf`, and you can find a list of its properties using `set(gcf)`. If you want to find the handle of a graphics object, click the mouse on the object and execute `gco`. You can then find any or all of the current properties of that object with command `get(gco)`.

The easiest way to MATLAB's online documentation for a complete treatment of handle graphics is by selecting **Help**→**Graphics Help**. Find the link to the Handle Graphics Property Browser. This gives an overview of all of the classes of graphics objects and their properties. It is an invaluable aid if you use handle graphics.

Experiment further with the various properties of the objects in the figure you created in Example 8.

## Exercises

In Exercises 1 – 6, find the solution to the indicated initial value problem, and use `ezplot` to plot it.

1.  $y' = -ty$  with  $y(0) = 1$  over  $[0, 2]$ .
2.  $y' = t(y + 1)$  with  $y(0) = 1$  over  $[0, 2]$ .
3.  $y' = -y + \cos t$  with  $y(0) = 2$  over  $[0, 5]$ .
4.  $y' = -y^2 \cos t$  with  $y(0) = 3$  over  $[0, 3]$ .
5.  $x' = -3x + t + e^{-2t}$  with  $x(0) = 0$  over  $[0, 4]$ .
6.  $z' = 3z + t^2 e^{-t}$  with  $z(0) = 1$  over  $[0, 2]$ .

In Exercises 7 – 10, the solutions are defined implicitly. Find the solution and plot it using `ezplot` in a region which displays the most important aspects of the solution. On the basis of your graph estimate the interval of existence.

7.  $y' = (1 + 3t^2)/(3y^2 - 6)$  with  $y(0) = 0$ .      8.  $(1 + 3y^2)y' = \cos(t)$  with  $y(0) = 1$ .  
 9.  $y' = 3 \sin t/(3y^2 - 4)$  with  $y(0) = 0$ .      10.  $y' = 3 \sin t/(3y^2 - 4)$  with  $y(0) = -2$ .

The `ezplot` command will also handle parametric equations. Try `ezplot('cos(t)', 'sin(t)', [0, 2*pi])` to get a feel for how the command works. In Exercises 11 – 14, use the `ezplot` command to plot the parametric equations over the indicated time interval.

11.  $x = \cos(2t) + 2 \sin(2t)$ ,  $y(t) = -\sin(2t)$ ,  $[0, 2\pi]$   
 12.  $x = \cos(t/2) + 2 \sin(t/2)$ ,  $y(t) = \sin(t/2)$ ,  $[0, 4\pi]$   
 13.  $x = e^{-t}(10 \cos(5t) + 20 \sin(5t))$ ,  $y(t) = 10e^{-t} \sin(5t)$ ,  $[0, 8\pi]$   
 14.  $x = e^t(\cos(4t) - \sin(4t))$ ,  $y(t) = 2e^t \sin(4t)$ ,  $[0, 8\pi]$

If the Symbolic Toolbox is installed in your MATLAB system, use the `dsolve` command to find the solution of the first order initial value problems in Exercises 15 – 18. Use the `ezplot` command to plot the solution over the indicated interval.

15.  $y' = -2ty$ ,  $y(0) = 1$ ,  $[-2, 2]$       16.  $y' + 2y = \cos(t)$ ,  $y(0) = 1$ ,  $[0, 20]$   
 17.  $y' = 1 + y^2$ ,  $y(0) = 1$ ,  $[-\pi, \pi]$       18.  $y' + y/t = ty^2$ ,  $y(2) = 3$ ,  $[-4, 6]$

In Exercises 19 – 24, find the solution to the given initial value problem. Write a script M-file to plot the solution over the indicated interval properly annotated with labels and a title.

19.  $y' + ty = y$  with  $y(1) = 3$  over  $[-2, 4]$   
 20.  $ty' = 2y + t^3 \cos(t)$  with  $y(\pi) = 0$  over  $[-2\pi, 2\pi]$   
 21.  $y' = y \sin(t)$  with  $y(0) = 1$  over  $[-2\pi, 2\pi]$   
 22.  $y' = ty^3$  with  $y(0) = -1$  over  $[-3/4, 3/4]$   
 23.  $y' + y \cos(t) = \cos(t)$  with  $y(\pi) = 0$  over  $[0, 4\pi]$   
 24.  $y' = y \cos(t)$  with  $y(0) = -1$  over  $[0, 6\pi]$   
 25. On the same figure plot  $y = \cos(x)$  and  $z = \sin(x)$  over the interval  $[0, 4\pi]$ . Use different line styles or colors for each curve, and label the figure appropriately.  
 26. On the same figure plot the three curves  $y = \sin(x)$ ,  $y = x - x^3/6$ , and  $y = x - x^3/6 + x^5/120$  over the interval  $[-3, 3]$ . Use different line styles or colors for each curve, and label the figure appropriately. Do you recognize the relationship between these three functions?  
 27. On the same figure plot the graphs of the function  $y = e^x$  and its Taylor approximations of order 1, 2, and 3 over the interval  $[-3, 3]$ . Use different line styles or colors for each curve, and label the figure appropriately.  
 28. Consider the functions  $y_1 = x$ ,  $y_2 = x^2$ , and  $y_3 = x^4$  on the interval  $[0.1, 10]$ . Plot these three functions on the same figure using the command `plot`. Now do the same thing with the other plotting commands `semilogx`, `semilogy`, and `loglog`. Turn in only the one that you think is most revealing about the relationship between these functions. Use different line styles or colors for each curve, and label the figure appropriately. (Plotting more than one curve on a figure using any of these commands follows the same procedure used with `plot`.)

For each set of parametric equations in Exercises 29 – 32, use a script file to create two plots. First, draw a plot of both  $x$  and  $y$  versus  $t$ . Use `handle graphics` to apply different linestyles and color to the plots of  $x$  and  $y$ , then add a legend, axis labels, and a title. Open a second figure window by placing the `figure` command at this point in your script, then draw a plot of  $y$  versus  $x$ . Add axis labels and a title to your second plot.

29.  $x = \cos(t) - 3 \sin(t)$ ,  $y = -2 \sin(t) - \cos(t)$ ,  $[0, 6\pi]$   
 30.  $x = \cos(2t) - 8 \sin(2t)$ ,  $y = -5 \sin(2t) - \cos(2t)$ ,  $[0, 4\pi]$   
 31.  $x = e^{-t}(\cos(2t) + 3 \sin(2t))$ ,  $y = e^{-t}(7 \sin(2t) - \cos(2t))$ ,  $[-2\pi, 2\pi]$   
 32.  $x = e^t(\cos(3t) - 3 \sin(3t))$ ,  $y = e^t(-2 \sin(3t) - \cos(3t))$ ,  $[-2\pi, 2\pi]$

33. In three dimensions plot the curve defined by

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

over the interval  $t \in [0, 4\pi]$  with the `plot3` command. Label the figure appropriately.

In Exercises 34 – 41, find the general solution of the differential equation. Then plot the family of solutions with the indicated initial values over the specified interval. We will use MATLAB notation to indicate the range of initial values. You can use the method of Example 7, but think about using a `for` loop.

34.  $y' + y = \sin t$  on the interval  $[0, 4\pi]$  with initial conditions  $y(0) = -10 : 2 : 10$ .  
 35.  $y' + y = 5$  on the interval  $[0, 4]$  with initial conditions  $y(0) = -10 : 2 : 10$ .  
 36.  $y' + \cos(x) \cdot y = \cos(x)$  on the interval  $[-10, 10]$  with initial conditions  $y(0) = -10 : 2 : 10$ .  
 37.  $y' = y - 3e^{-t}$  on the interval  $[-2, 2]$  with initial conditions  $y(0) = -5 : 1 : 5$ .  
 38.  $y' = y \cos t - 3y$  on the interval  $[0, 3]$  with initial conditions  $y(0) = -0.4 : 0.1 : 0.4$ .  
 39.  $y' = (1 + y^2) \cos t$  on the interval  $[0, 4\pi]$  with initial  $y(0) = -0.4 : 0.1 : 0.4$ .  
 40.  $2yy' = \cos t$  on the interval  $[0, \pi]$  with initial conditions  $y(\pi/2) = -3, -2, -1, 1, 2, 3$ .  
 41.  $(2 + 2y)y' = \sin t$  on the interval  $[0, 4\pi]$  with initial conditions  $y(0) = -3, -2, 0, 1, 2, 3$ .  
 42. The voltage across the capacitor in a driven  $RC$ -circuit is modeled by the initial value problem  $V_c' + V_c = \cos(t)$ ,  $V_c(0) = 0$ . The solution of the problem can be written  $V_c = V_t + V_s$ , where

$$V_s = \frac{1}{2} \cos(t) + \frac{1}{2} \sin(t) \quad \text{and} \quad V_t = -\frac{1}{2} e^{-t}.$$

The solution  $V_s$  is called the *steady-state solution* and the solution  $V_t$  is called the *transient solution*. On one plot, sketch the solutions  $V_s$ ,  $V_t$ , and  $V_c$  in blue, red, and black, respectively, over the time interval  $[0, 6\pi]$ . Add a legend to your plot.

43. Use the appropriate sum-to-product identity from trigonometry to show that

$$\sin(12t) - \sin(14t) = -2 \sin(t) \cos(13t).$$

On one plot, plot  $y = -2 \sin(t) \cos(13t)$  and its envelopes  $y = \pm 2 \sin(t)$  over the time interval  $[-2\pi, 2\pi]$ . Use the selection tool on the figure toolbar to select each envelope, then right-click the selected envelope to change both its color and linewidth.

44. Use the appropriate sum-to-product identity from trigonometry to show that

$$\cos(18t) - \cos(20t) = 2 \sin(t) \sin(19t).$$

On one plot, plot  $y = 2 \sin(t) \sin(19t)$  and its envelopes  $y = \pm 2 \sin(t)$  over the time interval  $[-2\pi, 2\pi]$ . Using handle graphics in a script file, change the color and linewidth of the envelopes.