

4.

The Use of Function M-Files

In Chapter 2 we discussed script M-files, and how they can be used to facilitate the preparation of complicated graphics. Here we will introduce *function M-files*, which are very like script M-files, but can pass parameters and isolate variables. These two capabilities can be exploited to add new functions to MATLAB's already extensive list, and to extend our ability to perform computational experiments and prepare graphics.

In this chapter we will emphasize the use of MATLAB's built-in editor. If you use a different editor, and one of the authors does, then ignore these parts. However, the MATLAB editor has several connections with MATLAB itself that make it a good choice.

New Functions in MATLAB

It is very easy to add your own functions to the long list provided by MATLAB. Let's start with a very easy example.

Example 1. Create a function M-file for the function $f(x) = x^2 - 1$.

Execute `edit` at the command line to open the MATLAB editor¹ and enter the following three lines (one of them is blank, and is provided for readability):

```
function y = f(x)

y = x^2-1;
```

The first line of a function M-file must conform to the indicated format. It is this first line that distinguishes between a function M-file and a script M-file. The very first word must be the word `function`. The rest of the first line has the form

```
dependent_variables = function_name(independent_variables)
```

In the function `f`, `y = f(x)` indicates that x is the independent variable and y is the dependent variable.

The rest of the function M-file defines the function using the same syntax we have been using at the MATLAB prompt. Remember to put a semicolon at the end of lines in which computations are done. Otherwise the results will be printed in the Command Window.

When you save the file, you will be prompted to use the file name `f.m`. The file name should always be the same as the function name, followed by the suffix `.m`, so accept the suggestion. That's all there is to it. Now if you want to compute $f(3) = (3)^2 - 1 = 8$, simply enter `f(3)` at the MATLAB prompt.

```
>> f(3)
ans =
     8
```

¹ There are several ways to open the editor. Explore the Toolbar and the **Edit** menu.

Making Functions Array Smart. There is one important enhancement you will want to make to the M-file `f.m`. If you try to compute f on a matrix or a vector, you will find that it is not array smart.

```
>> x = 1:5
x =
     1     2     3     4     5
>> f(x)
??? Error using ==> ^
Matrix must be square.
```

This error message refers to the fact that we tried to compute x^2 for a vector x . We forgot to use array exponentiation instead of ordinary exponentiation. To edit the function to make it array smart we simply add one period:

```
function y = f(x)

y = x.^2-1;
```

Now the function can handle matrices. With the same vector x , we get

```
>> f(x)
ans =
     0     3     8    15    24
```

Notice that if x is a matrix, then so is $x.^2$. On the other hand, 1 is a number, so the difference $x.^2-1$ is not defined in ordinary matrix arithmetic. However, MATLAB allows it, and in cases like this will subtract 1 from every element of the matrix. This is a very useful feature in MATLAB.

Example 2. *An object thrown in the air obeys the initial value problem*

$$\frac{d^2y}{dt^2} = -9.8 - \frac{dy}{dt}, \quad \text{with } y(0) = 0 \quad \text{and} \quad y'(0) = 120.$$

When we solve this linear equation, we find that

$$y = -\frac{49}{5}t + \frac{649}{5}(1 - e^{-t}),$$

where y is the height in meters of the object above ground level after t seconds. Estimate the height of the ball after 5 seconds.

This is a perfect situation for a function M-file, particularly if you are interested in predicting the height of the ball at a number of additional times other than $t = 5$ seconds. Open a new M-file in the editor, enter

```
function y = height(t)
y = -(49/5)*t + (649/5)*(1 - exp(-t));
```

and save it as `height.m`. This time we needed no additional periods to make the function array smart. It is now a simple matter to find the height at $t = 5$ seconds.

```
>> y = height(5)
y =
    79.9254
```

Hence, the height of the ball at $t = 5$ seconds is 79.9254 meters.

Example 3. *Plot the height of the object in Example 2 versus time and use your graph to estimate the maximum height of the object and the time it takes the object to return to ground level.*

Although we could operate strictly from the command line to obtain a plot, we will use a script M-file to do the work for us. Open a new M-file, enter the commands

```
close all
t = linspace(0,15,200);
y = height(t);
plot(t,y)
grid on
xlabel('time in seconds')
ylabel('height in meters')
title('Solution of y'''' = -9.8 - y'', y(0) = 0, y''(0) = 120')
```

and save it in a file named `height_drv.m`. Notice that, since the first line does not begin with the word `function`, this is a script M-file, not a function M-file.

The command `close all` will close all open figure windows. The purpose of this command is to prevent figure windows from accumulating as we execute versions of the script. However, you should be cautious about using it, since you may close figures you want to have open. Notice that we used the function `height` defined in Example 2 to calculate the y -values over the time interval $[0, 15]$.

You can execute the script M-file by typing `height_drv` at the MATLAB prompt. However, you can also select **Debug**→**Run**² from the editor menu. This menu item has the accelerator key F5. This means that after you edit the file, you can both save it and execute it with F5. The routine of editing, followed by F5 can be a great time saver as you refine an M-file.

Running your script M-file should produce an image similar to that shown in Figure 4.1. If your file contain errors, you might hear a “beep” or a “click” when you press the F5 button. An error message will be reported in the MATLAB command window, often as a link. Clicking this link will bring up the editor with the cursor on the line in the where the error occurs.

By examining the graph, we see that the object reaches a maximum height between 90 and 100 meters after about two or three seconds of flight. It returns to ground level (height zero) after approximately thirteen or fourteen seconds. You can use the zoom tool located on the toolbar in the figure window to obtain better estimates.

² This menu item changes to **Save and Run** if you have made changes since the last time you saved the file.

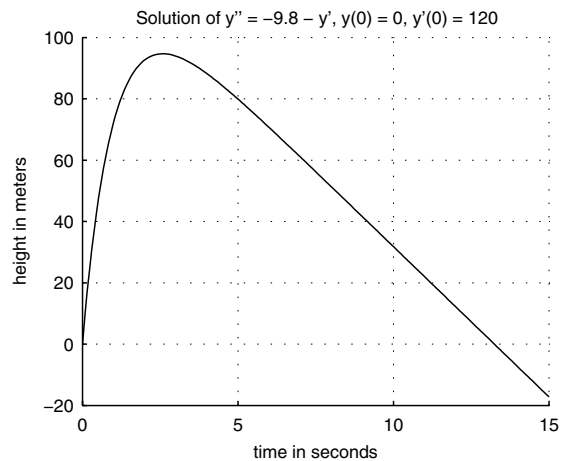


Figure 4.1. Plotting the solution of $y'' = -9.8 - y'$, $y(0) = 0$, $y'(0) = 120$.

Function M-files, Computational Exercises and Graphics

Some of you might wonder why we had to use *both* the function M-file `height.m` and the script M-file `height_drv.m` to produce Figure 4.1 in Example 2. It takes time to write files, and the accumulation of files clutters up the directory in which they are saved. Starting with version 6 of MATLAB there is a way around this problem.

Subfunctions in MATLAB 6. The answer is to define the function `height` as a *subfunction* with the file `height_drv.m`. However, subfunctions are not allowed in script M-files.

Open MATLAB's built-in editor, enter

```
function ch4examp3
close all
t = linspace(0,15,200);
y = height(t);
plot(t,y)
grid on
xlabel('time in seconds')
ylabel('height in meters')
title('Solution of y'''' = -9.8 - y'', y(0) = 0, y''(0) = 120')

function y = height(t)
y = -(49/5)*t + (649/5)*(1 - exp(-t));
```

and save it as `ch4examp3.m`. Use the F5 key to execute the file. This should produce a plot similar to that in Figure 4.1.

Notice that the code in `ch4examp3` is precisely the same as the code in the two files `height.m` and `height_drv.m` created in Examples 2 and 3. However, this is a single function M-file with `height` included as a subfunction.

Functions of functions — MATLAB’s funfun directory. MATLAB has a large number of functions that act on functions. If you type

```
>> help funfun
```

at MATLAB’s prompt, you will see a list of MATLAB routines for finding zeros and extrema of functions, tools for numerical integration, and many others. There is a suite of routines for solving differential equations, which we will look at in Chapter 8. For help on any of these functions, say `fzero`, execute `help fzero` and learn how to find the zeros of a function.

Example 4. Use `fzero` to find the time it takes the object in Example 2 to return to ground level.

The help command indicates that “`X = FZERO(FUN,X0)` tries to find a zero of the function `FUN` near `X0`.” Thus, `fzero` needs two inputs, the name of a function, and an approximate value of the zero. We will use the function `height` from Example 2, and from Figure 4.1 we estimate that `height` has a zero near $t = 13$. In all versions of MATLAB you can invoke `fzero` with the command

```
>> t = fzero('height',13)
t =
    13.2449
```

Notice that the function name must be entered between single quotes. Starting with version 6 of MATLAB, the use of

```
>> t = fzero(@height,13)
t =
    13.2449
```

is encouraged. The expression `@height` is a *function handle* for the function `height`. In either case, we find that the object returns to ground level after about 13.2 seconds. This certainly agrees with that we see in Figure 4.1.

Example 5. Find the maximum height reached by the object in Example 2.

In the list provided by `help funfun` we find that MATLAB doesn’t provide a function for finding the maximum of a function. However, the routine `fminbnd` will find a local minimum, and a local maximum of a function can be determined by finding a local minimum of the negative of the function.

Using `help fminbnd`, we learn that finding the minimum of a function on a given interval requires the input of a function and the beginning and endpoints of the interval in which to find the minimum. Consequently, we will apply `fminbnd` to the negative of `height` on the interval $[0, 5]$. One approach would be to write a new function M-file encoding the negative of $-(49/5)t + (649/5)(1 - e^{-t})$. However, let’s pursue an alternative.

MATLAB provides a perfect solution for the situation when you need a function, but do not want to go to the trouble of writing a function M-file, called an *inline function*.³ We could create our inline

³ Type `help inline` to get a thorough explanation of the use of inline functions.

function for the negative of `height` as

```
>> f = inline('(49/5)*t - (649/5)*(1 - exp(-t))','t')
f =
    Inline function:
    f(t) = (49/5)*t - (649/5)*(1 - exp(-t))
```

However, since we have already created `height`, it is easier to use the inline function

```
>> f = inline('-height(t)','t')
f =
    Inline function:
    f(t) = -height(t)
```

From Figure 4.1 we estimate that the maximum occurs between $t = 0$ and $t = 5$, so the command

```
>> t = fminbnd(f,0,5)
t =
    2.5836
```

finds the time at which the maximum occurs. Notice that there are no single quotes around `f` in `fminbnd(f,1,5)`, so inline functions are treated differently from function M-files as inputs. Alternatively, the syntax `t = fminbnd(inline('-height(t)'),1,5)` will do the job in a single command.

The maximum height is now easily calculated by

```
>> h = -f(t)
h =
    94.6806
```

or by `h = height(t)`. Compare these results with Figure 4.1.

Using function M-files to pass parameters. The ability of function M-files to have input variables can be used to advantage when producing graphics. We will present a simple example.

Example 6. Write a function M-file that will plot the solution to the initial value problem

$$y' + y = 2t \cos t \quad \text{with} \quad y(0) = y_0$$

over the interval $[0, 4\pi]$. Write the file with y_0 as an input variable.

The equation is linear, and we find that the general solution is $y(t) = t(\cos t + \sin t) - \sin t + Ce^{-t}$. At $t = 0$, we find that $y_0 = y(0) = C$. Thus the function M-file

```
function ch4examp6(y0)
t = linspace(0,4*pi);
y = t.*(cos(t) + sin(t)) - sin(t) + y0*exp(-t);
plot(t,y), shg
```

will produce the required graph when `ch4examp6(y0)` is executed at the command line. Of course, you will want to add formatting commands to your function M-file. The result is shown in Figure 4.2. The one tricky formatting command is the one that produces the title. The command

```
title(['y' + y = 2tcos(t) with y(0) = ', num2str(y0), '.'])
```

requires the concatenation of text strings, as explained in the section "Text Strings in MATLAB" in Chapter 2.

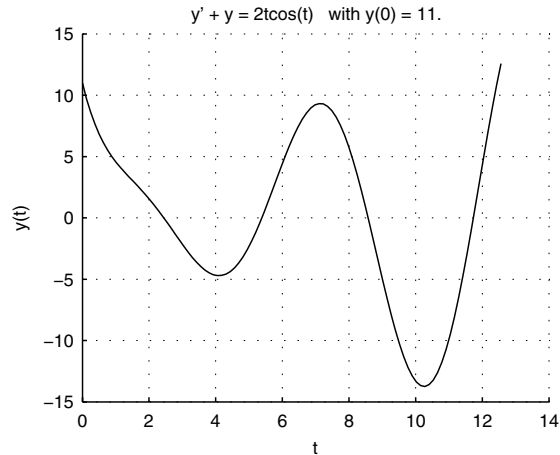


Figure 4.2. The output of `ch4examp6(11)`.

Functions of Several Variables

MATLAB allows multiple input and output variables in function M-files.

Example 7. Write a function M-file for the function defined by $f(t, x) = x^2 - t$.

We need to choose a name for this function, so let's use the mnemonic `xsqmt` ("x squared minus t"). Then the following M-file describes this function:

```
function y = xsqmt(t,x)
% This is Example 7 of Chapter 4.
y=x.^2-t;
```

The first line of the file starts with the word `function`. After that comes `y=xsqmt(t,x)`, meaning that `y` is the dependent variable, and both `t` and `x` are independent variables.

Everything on a line in an M-file after a percentage sign (`%`) is ignored by MATLAB. This can be utilized to put comments in a file, as we have done in the function `xsqmt`. Comment lines are useful ways to document your M-files. It is amazing how quickly we forget why we did things the way we did. Comment lines immediately after the function statement can be read using the MATLAB `help` command.

For example, if `xsqmt` is defined as above, saved as `xsqmt.m`, then `help xsqmt` gives the following response.

```
>> help xsqmt
```

```
This is Example 7 of Chapter 4.
```

You can evaluate the function at $(t, x) = (2, 5)$ as follows.

```
>> xsqmt(2,5)
ans =
    23
```

Naming and Organizing Files

Once you learn how to write M-files, they tend to accumulate rapidly. It is a good idea to give them appropriate names and to organize them into a directory structure that will make them easy to find. Of course you will choose a method that suits you. We will present some general considerations.

As one starts to use files, it is typical to save them all to the same directory. That is fine if you only have a few. When the number gets large enough that you have trouble identifying them, it becomes necessary to create new directories, and it is a good idea to use the directory system to differentiate the files. For example, you might create a directory named `manual` in which you save all files related to this Manual.

The method of creating directories or folders depends on your operating system. You can use the MATLAB command `mkdir`, but this is usually less convenient. Once you have more than one directory, you will need to know how to navigate through them. For one thing, MATLAB has a “current directory” in which it operates. This directory is displayed in the Current Directory popup menu on the toolbar of the command window. You can also discover the current directory by executing `pwd`. The popup menu lists all of the directories you have visited recently, making it easy to navigate among them. You can change to a directory not on the popup menu using the browse button, labeled with an ellipsis (`...`). You can also use the command `cd`.

The MATLAB path. After you have created a couple of directories, you will have to become familiar with the MATLAB path. Difficulties will arise in one of two ways. The first happens when you try to execute an M-file and get the error message

```
>> phony
??? Undefined function or variable 'phony'.
```

However, you are sure you created `phony.m` just yesterday, and it worked fine. The problem here is probably that `phony.m` is in a directory that is not on the MATLAB path. The second difficulty that can arise is that you execute an M-file, and, while it works, it is not doing what you know it should. In this case you probably have two files named `phony.m`, in different directories, and MATLAB is finding the wrong one.

The MATLAB path is the list of directories that MATLAB searches when it is looking for an M-file, in the order that they are searched. You can find the path using the `pathtool`,⁴ which can be accessed using the command `pathtool` or by choosing the menu item **File**→**Set Path...**. The first directory on the path is always the current directory, so you can be sure you are executing the correct M-file if you find out what directory it is in, and then change directories so that it is the current directory. It is also possible to add directories to the MATLAB path using the `pathtool`. Files that you use frequently should be in directories that are permanently on the path.

If you find that you have two files with the same name, say `phony`, you can find out which one is higher on the path with the command `which phony`. MATLAB will respond with the complete address of the first instance of `phony.m` on the MATLAB path. If the address is not the one you expect, you know that you have a name conflict. You could delete the offending file (probably not a good idea), you could change the name of one of the files, or you could just change directories so that the directory containing the right version of `phony.m` is the current directory.

Either of the commands `ls` or `dir` will display a list of the files in the current directory. You can display the contents of the M-file `phony.m` on the command window with the command `type phony`. This works very well with short files, but not too well with long ones.

Naming Functions and Variables. All of us typically refer to functions with one letter names, such as *f* or *g*. However, many important and commonly used functions have longer names. Think of the trigonometric functions, the logarithm, and the hyperbolic functions as examples. It is a good idea to follow that practice when naming your MATLAB M-files. Giving your function M-files distinctive names helps to avoid name conflicts and might help you remember their functionality at a later date. For example, in Example 3 the name `ch4examp3` was carefully chosen to reflect the fact that the file is related to Example 3 in Chapter 4 of this Manual.

The name of a function M-file has the form `function_name.m`, where `function_name` is the name you choose to call the function. While this name can be almost anything, there are a few rules.

- The name must start with a letter (either upper case or lower case).
- The name must consist entirely of letters, numerals, and underscores (`_`). No other symbols are allowed. In particular, no periods are allowed.
- The name can be arbitrarily long, but MATLAB will only remember the first 31 characters.
- Do not use names already in use such as `cos`, `plot`, or `dfield6`. If you do, MATLAB will not complain, but you will eventually suffer from the name duplication problem described earlier.

The variable names used in a function M-file must satisfy the same rules that apply to the names of M-files. Other than that they are arbitrary. As a result the file `f.m` of Example 1 could have been written as

```
function stink = funn(skunk)
stink = skunk.^2 - 1;
```

Save this file as `funn.m` and with `x = 1:5` execute

⁴ Or you can type `path` at the MATLAB prompt.

```
>> funn(x)
ans =
    0     3     8    15    24
```

Compare this to the output produced earlier with $f(x)$ and note that there is no difference. It is important to realize that the variable names used in function M-files are *local* to these files; i.e., they are not recognized outside of the M-files themselves. See Exercises 12 and 13. In addition variables that have been defined at the command line are not available within function M-files. Script M-files, on the other hand, do not isolate variables.

Exercises

Find the solution of each of the initial value problems in Exercises 1–4, then craft an inline function for the solution. After insuring that your inline function is “array smart,” use the inline function to (i) plot the solution of the initial value problem on the given interval, and (ii) evaluate the solution at the right endpoint of the given interval.

1. $y' = -t/y$, with $y(0) = 4$, on $[0, 4]$.
2. $y' = -2ty^2$, with $y(0) = 1$, on $[0, 5]$.
3. $y' + y = \cos t$, with $y(0) = 1$, on $[0, 5\pi]$.
4. $y' + 2y = \sin(2t)$, with $y(0) = 1$, on $[0, 4\pi]$.

Find the solution of each of the initial value problems in Exercises 5–8, then craft a function M-file for the solution. After insuring that your function M-file is “array smart,” use the function to (i) plot the solution of the initial value problem on the given interval, and (ii) evaluate the solution at the right endpoint of the given interval.

5. $y' = (1 - t)y$, with $y(0) = 1$, on $[0, 4]$.
6. $y' = y \sin(2t)$, with $y(0) = 1$, on $[0, 2\pi]$.
7. $y' = y/t + t$, $y(1) = -1$, on $[0, 3]$.
8. $y' + t^2y = 3$, with $y(0) = -2$, on $[0, 5]$.

Find the solution of the given initial value problem in Exercises 9–12, then craft a function M-file for the solution. After insuring that your function M-file is “array smart,” plot the solution on the given interval. Use MATLAB’s `fminbnd` function to find the minimum value of the solution on the given interval and the time at which it occurs. Use the figure window’s “Insert Text” and “Insert Arrow” tools to annotate your plot with these findings.⁵

9. $y' + 2ty = -e^{-t^2}$, with $y(0) = 0$, on $[0, 2]$.
10. $y' - y/t = -t \sin t$, with $y(\pi) = -\pi$, on $[0, 2\pi]$.
11. $y' = 1 + 2y/t$, with $y(1) = -3/4$, on $[1, 4]$.
12. $y' + y = t^2$, with $y(0) = 1$, on $[0, 2]$.

Find the solution of the given initial value problem in Exercises 13–16, then craft a function M-file for the solution. After insuring that your function M-file is “array smart,” plot the solution on the given interval. Use MATLAB’s `fminbnd` function and the technique of Example 5 to find the maximum value of the solution on the given interval and the time at which it occurs. Use the figure window’s “Insert Text” and “Insert Arrow” tools to annotate your plot with these findings.

13. $ty' - y = t^2 \cos(t)$, with $y(\pi) = 0$, on $[0, 2\pi]$.
14. $y' = y \cos(t)$, with $y(0) = 1$, on $[0, 2\pi]$.
15. $y' = (1 + y) \sin(t)$, with $y(\pi) = 0$, on $[0, 2\pi]$.

⁵ If the figure toolbar is not visible, select **View** → **Figure Toolbar** from the figure menu.

16. $y' + t^2y = 3$, with $y(0) = 1$, on $[0, 3]$.
17. The differential equation $y' + y = \cos(4t)$ has the general solution $y = (1/17)\cos(4t) + (4/17)\sin(4t) + Ce^{-t}$. Craft a function M-file for this solution as follows.

```
function y = f(t,C)
y = 1/17*cos(4*t) + 4/17*sin(4*t) + C*exp(-t);
```

Plot a *family of solutions* for the differential equation with the following script.

```
t = linspace(0,2*pi,500);
y = [];
for C = -20:5:20
    y = [y;f(t,C)];
end
plot(t,y)
```

Follow the lead of Exercise 17 to plot the given families of solutions in Exercises 18–23. Use the indicated constants and extend your plot over the given time interval.

18. $y = 2/(t^2 + C)$, $C = 1, 2, 3, 4, 5$, on $[-2, 2]$.
19. $y = (1/4)t^2 - (1/8)t + 1/32 + Ce^{-4t}$, $C = 1, 2, 3, 4, 5$, on $[0, 2]$.
20. $y = Cte^{-t^2}$, $C = -3, -2, -1, 0, 1, 2, 3$, on $[-2, 2]$.
21. $y = (t/2)(\cos(t) + \sin(t)) - (1/2)\sin(t) + Ce^{-t}$, $C = -3, -2, -1, 0, 1, 2, 3$, on $[0, 10]$.
22. $y = -1 + \cos(t) + (C + t/2)\sin(t)$, $C = 0, 1, 2, 3, 4, 5$, on $[0, 6\pi]$.
23. $y = (1/10)\sin(t) + (1/5)\cos(t) + e^{-t}(C\sin(2t) - (1/5)\cos(2t))$, $C = -3, -2, -1, 0, 1, 2, 3$, on $[0, 2\pi]$.
24. A tank initially contains 20 gallons of pure water. A salt solution containing 2 lb/gal flows into the tank at a rate of 3 gal/min. A drain is open at the bottom of the tank through which flows salt solution from the tank at a rate of 2 gal/min. Set up and solve an initial value problem modeling the amount of salt in the tank at time t minutes. Write an inline function for your solution and use it to find the salt content in the tank at $t = 30$ minutes.
25. Tank A initially contains 40 gallons of a salt solution. Pure water is poured into tank A at a rate of 2 gal/min. A drain is opened at the bottom of tank A so that salt solution flows directly from tank A into tank B at 2 gal/min, keeping the volume of the salt solution in tank A constant over time. Tank B initially contains 100 gallons of pure water. A drain is opened at the bottom of tank B so that the volume of solution in tank B remains constant over time. If x and y represent the salt content in tanks A and B, respectively, show that

$$\frac{dx}{dt} = -\frac{x}{25} \quad \text{and} \quad \frac{dy}{dt} = \frac{x}{25} - \frac{y}{50}.$$

Given that $x(0) = 20$, solve the first equation for x , substitute the result into the second equation, then show that

$$y = 40e^{-t/50} - 40e^{-t/25}.$$

Write a function M-file to evaluate y at time t . Following the technique used in Examples 3-5, use your function to (i) construct a plot of the salt content in tank B over the time interval $[0, 100]$, and (ii) use `fminbnd` to find the maximum salt content in tank B and the time at which this occurs.

26. An object thrown into the air is known to obey the initial value problem

$$y'' = -9.8 - 0.05y', \quad y(0) = 0, \quad y'(0) = 200,$$

where y is the height of the ball above ground (in meters) at t seconds. The solution of this initial value problem is

$$y(t) = -196t + 7920(1 - e^{-t/20}).$$

Following the lead of Examples 2-5 in the text, plot this solution, then find its maximum height and the time at which it occurs. Then find the time it takes the object to return to the ground. Question: Does the time it takes the object to reach its maximum height equal the time it takes the object to return to ground level from this maximum height? In other words, does it take the object the same amount of time to go up as it takes the object to come down?

27. A simple RC -circuit with emf $V(t) = 3 \cos(\omega t)$ is modeled by the initial value problem

$$RCV'_C + V_C = 3 \cos(\omega t), \quad V_C(0) = 0,$$

where R is the resistance, C the capacitance, ω is the driving frequency, and V_C is the voltage response across the capacitor. Show that the solution is

$$V_C(t) = \frac{RC\omega \sin \omega t + \cos \omega t - e^{-t/(RC)}}{1 + R^2C^2\omega^2}.$$

Create a function M-file for this solution as follows.

```
function V = f(t,R,C,w)
V = (R*C*w*sin(w*t) + cos(w*t) - exp(-t/(R*C)))/(1 + R^2*C^2*w^2);
```

Now, call this function with the following script.

```
R = 1.2; C = 1; w = 1;
t = linspace(0,2*pi,1000);
V = f(t,R,C,w);
plot(t,V)
```

Run this script for $\omega = 1, 2, 4,$ and $8,$ keeping $R = 1.2$ ohms and $C = 1$ farad constant. What happens to the amplitude of the voltage response across the capacitor as the frequency of the emf is increased? Why do you think this circuit is called a *low pass filter*?

28. Write a function M-file to calculate $f(z) = \sqrt{|z|}$. Use it to graph f over the interval $(-3, 2)$.
29. Write a function M-file to compute the function $f(t) = t^{1/3}$. Use it to graph the function over the interval $[-1, 1]$. (This is not as easy as it looks. Read the section on complex arithmetic in Chapter 1.)
30. Write a function M-file to calculate the function $f(t) = e^{-t}(t^2 + 4e^t - 5)$.
- Use the M-file to graph f over the interval $(-2.5, 3)$.
 - Find all of the zeros in the interval $(-2.5, 3)$ of the function f defined in the previous exercise.
31. For the function `funn` defined in this chapter find all solutions to the equation $\text{funn}(x) = 5$ in the interval $[-5, 5]$.
32. Write an array smart function M-file to compute $f(t) = 5$. **Remark:** You will find this exercise more difficult than it looks. There are a variety of tricks that will work, but the methods that are most consistent with the ways used for non constant functions (and therefore are most useful in programming applications) use the MATLAB commands `size` and `ones`. Use `help` on these commands.
33. Variables defined in functions are *local* to the function. To get a feel for what this means, create the function M-file

```
function y = fcn(x)
A = 2;
y = A^x;
```

Save the file as `fcn.m`. In MATLAB's command window, enter the following commands.

```
A = 3; x = 5;
y = fcn(x)
A^x
```

Explain the discrepancy between the last two outputs. What is the current value of A in MATLAB's workspace?

34. You can make variables *global*, allowing functions access to the variables in MATLAB's workspace. To get a feel for what this means, create the function M-file

```
function y = gcn(x)
global A
A = 2;
y = A^x;
```

Save the file as gcn.m. In MATLAB's command window, enter the following commands.

```
global A
A = 3; x = 5;
y = gcn(x)
A^x
```

Why are these last two outputs identical? What is the current value of A in MATLAB's workspace?