Copyrighted Materials
Copyright © 2010 Elsevier Retrieved from www.knovel.com

CHAPTER 3

# Program Design and Algorithm Development

The objectives of this chapter are to introduce you to

- Program design
- The structure plan (pseudo-code) as a means of designing the logic of a program

This chapter is an introduction to the design of computer programs. The top-down design process is elaborated to help you think about good problem-solving strategies as they relate to the design of procedures for using software like MATLAB. We will consider the design of your own toolbox to be included among those already available with your version of MATLAB, such as Simulink, Symbolic Math, and Controls System. This is a big advantage of MATLAB (and tools like it); it allows you to customize your working environment to meet your own needs. It is not only the "mathematics handbook" of today's student, engineer, and scientist, it is also a useful environment to develop software tools that go beyond any handbook to help you to solve relatively complicated mathematical problems.

In the first part of this chapter we discuss the design process. In the second part we examine the structure plan—the detailed description of the algorithm to be implemented. We will consider relatively simple programs. However, the process described is intended to provide insight into what you will confront when you deal with more complex engineering, scientific, and mathematical problems during the later years of your formal education, your life-long learning, and your continuing professional education.

To be sure, the examples examined so far have been logically simple. This is because we have been concentrating on the technical aspects of writing correct

Copyright © 2010, Daniel T. Valentine. Published by Elsevier Ltd. All rights reserved.
DOI: 10.1016/B978-0-12-374883-6.00003-5

MATLAB statements. It is very important to learn how MATLAB does the arithmetic operations that form the basis of more complex programs. To design a successful program you need to understand a problem thoroughly and break it down into its most fundamental logical stages. In other words, you have to develop a systematic procedure or *algorithm* for solving it.

There are a number of methods that may assist in algorithm development. In this chapter we look at one, the *structure plan*. You briefly met the concept of a structure plan. Its development is the primary part of the software (or code) design process because it is the steps in it that are translated into a language the computer can understand—for example, into MATLAB commands, some of which were introduced in the previous two chapters.

## 3.1 THE PROGRAM DESIGN PROCESS

Useful utilities translated into MATLAB (either sequences of command lines or *functions*, which are described later in the text) and saved as M-files in your working directory are your primary goals. There are numerous toolboxes available through MathWorks (among others) on a variety of engineering and scientific topics. A great example is the Aerospace Toolbox, which provides reference standards, environmental models, and aerodynamic coefficient importing for advanced aerospace engineering designs. Explore the MathWorks Web site for products available (http://www.mathworks.com/).

In the default working directory, `\work`, or in your own working directory (e.g., `\mytools`), you will begin to accumulate a set of M-files that you have created as you use MATLAB. One way to create and to get to your own working directory is to execute the following commands:

```
>> mkdir mytools   <Enter>
>> cd mytools      <Enter>
```

Certainly, you want to be sure that the tools you save are reasonably well written (i.e., reasonably well designed). What does it mean to create well-written programs?

The goals in designing a software tool are that it works, it can easily be read and understood, and, hence, it can be systematically modified when required. For programs to work well they must satisfy the requirements associated with the problem or class of problems they are intended to solve. The specifications (i.e., the detailed description of purpose, or function, inputs, method of processing, outputs, and any other special requirements) must be known to design an effective algorithm or computer program, which must work completely and correctly. That is, all options should be usable without error within the limits of the specifications.

The program must be readable and hence clearly understandable. Thus, it is useful to decompose major tasks (or the main program) into subtasks (or subprograms) that do specific parts of it. It is much easier to read subprograms, which have fewer lines, than one large main program that doesn't segregate the subtasks effectively, particularly if the problem to be solved is relatively complicated. Each subtask should be designed so that it can be evaluated independently before it is implemented in the larger scheme of things (i.e., in the main program plan).

A well written code, when it works, is much more easily evaluated in the testing phase of the design process. If changes are necessary to correct sign mistakes and the like, they can be easily implemented. One thing to keep in mind when you add comments to describe the process programmed is this: Add enough comments and references so that a year from the time you write the program you know exactly what was done and for what purpose. Note that the first few comment lines in a script file are displayed in the Command Window when you type `help` followed by the name of your file (file naming is also an art).

The design process[1] is outlined next. The steps may be listed as follows:

**Step 1** *Problem analysis.* The context of the proposed investigation must be established to provide the proper motivation for the design of a computer program. The designer must fully recognize the need and must develop an understanding of the nature of the problem to be solved.

**Step 2** *Problem statement.* Develop a detailed statement of the mathematical problem to be solved with a computer program.

**Step 3** *Processing scheme.* Define the inputs required and the outputs to be produced by the program.

**Step 4** *Algorithm.* Design the step-by-step procedure in a *top-down* process that decomposes the overall problem into subordinate problems. The subtasks to solve the latter are refined by designing an itemized list of steps to be programmed. This list of tasks is the *structure plan* and is written in *pseudo-code* (i.e., a combination of English, mathematics, and anticipated MATLAB commands). The goal is a plan that is understandable and easily translated into a computer language.

**Step 5** *Program algorithm.* Translate or convert the algorithm into a computer language (e.g., MATLAB) and debug the syntax errors until the tool executes successfully.

**Step 6** *Evaluation.* Test all of the options and conduct a validation study of the program. For example, compare results with other programs that

[1]For a more detailed description of software design technology see, for example, *C++ Data Structures* by Nell Dale (Jones and Bartlett, 1998).

do similar tasks, compare with experimental data if appropriate, and compare with theoretical predictions based on theoretical methodology related to the problems to be solved. The objective is to determine that the subtasks and the overall program are correct and accurate. The additional debugging in this step is to find and correct *logical* errors (e.g., mistyping of expressions by putting a plus sign where a minus sign was supposed to be) and *runtime* errors that may occur after the program successfully executes (e.g., cases where division by zero unintentionally occurs).

**Step 7** *Application.* Solve the problems the program was designed to solve. If the program is well designed and useful, it can be saved in your working directory (i.e., in your user-developed toolbox) for future use.

### 3.1.1 The projectile problem

*Step 1.* Let us consider the projectile problem examined in first-semester physics. It is assumed that engineering and science students understand this problem (if it is not familiar to you, find a physics text that describes it or search the Web; the formulas that apply will be provided in step 2).

In this example we want to calculate the flight of a projectile (e.g., a golf ball) launched at a prescribed speed and a prescribed launch angle. We want to determine the trajectory of the flight path and the horizontal distance the projectile (or object) travels before it hits the ground. Let us assume zero air resistance and a constant gravitational force acting on the object in the opposite direction of the vertical distance from the ground. The launch angle, $\theta_o$, is defined as the angle measured from the horizontal (ground plane) upward toward the vertical direction, $0 < \theta_o \leq \pi/2$, where $\theta_o = 0$ implies a launch in the horizontal direction and $\theta_o = \pi/2$ implies a launch in the vertical direction (i.e., in the opposite direction of gravity). If $g = 9.81$ m/s$^2$ is used as the acceleration of gravity, the launch speed, $V_o$, must be entered in units of m/s. Thus, if the time, $t > 0$, is the time in seconds (s) from the launch time of $t = 0$, the distance traveled in $x$ (the horizontal direction) and $y$ (the vertical direction) is in meters (m).

We want to determine the time it takes the projectile, from the start of motion, to hit the ground, the horizontal distance traveled, and the shape of the trajectory. In addition, we want to plot the speed of the projectile versus the angular direction of this vector. We need, of course, the theory (or mathematical expressions) that describes the solution to the zero-resistance projectile problem in order to develop an algorithm to obtain solutions to it.

*Step 2.* The mathematical formulas that describe the solution to the projectile problem are provided in this step. Given the launch angle and launch speed, the horizontal distance traveled from $x = y = 0$, which is the coordinate location of the launcher, is

$$x_{max} = 2\frac{V_o^2}{g}\sin\theta_o\cos\theta_o$$

The time from $t = 0$ at launch for the projectile to reach $x_{max}$ (i.e., its range) is

$$t_{x_{max}} = 2\frac{V_o}{g}\sin\theta_o$$

The object reaches its maximum altitude,

$$y_{max} = \frac{V_o^2}{2g}\sin^2\theta_o$$

at time

$$t_{y_{max}} = \frac{V_o}{g}\sin\theta_o$$

The horizontal distance traveled when the object reaches the maximum altitude is $x_{y_{max}} = x_{max}/2.0$.

The trajectory (or flight path) is described by the following pair of coordinates at a given instant of time between $t = 0$ and $t_{x_{max}}$:

$$x = V_o t\cos\theta_o$$

$$y = V_o t\sin\theta_o - \frac{g}{2}t^2$$

We need to solve these equations over the range of time $0 < t \le t_{x_{max}}$ for prescribed launch conditions $V_o > 0$ and $0 < \theta \le \pi/2$. Then the maximum values of the altitude and the range are computed along with their respective arrival times. Finally, we want to plot $V$ versus $\theta$, where

$$V = \sqrt{(V_o\cos\theta_o)^2 + (V_o\sin\theta_o - gt)^2}$$

and

$$\theta = \tan^{-1}\left(\frac{V_y}{V_x}\right)$$

We must keep in mind when we study the solutions based on these formulas that the air resistance was assumed negligible and the gravitational acceleration was assumed constant.

*Step 3*. The required inputs are $g$, $V_o$, $\theta_o$, and a finite number of time steps between $t = 0$ and the time the object returns to the ground. The outputs are the range and time of flight, the maximum altitude and the time it is reached, and the shape of the trajectory in graphical form.

*Steps 4 and 5*. The algorithm and structure plan developed to solve this problem are given next as a MATLAB program, because it is relatively straightforward and the translation to MATLAB is well commented with details of the approach applied to its solution (i.e., the steps of the structure plan are enumerated). This plan, and M-file, of course, is the summary of the results developed by trying a number of approaches during the design process, and thus discarding numerous sheets of scratch paper before summarizing the results! (There are more explicit examples of structure plans for your review and investigation in the next section of this chapter.) Keep in mind that it was not difficult to enumerate a list of steps associated with the general design process, that is, the technical problem solving. However, it is certainly not so easy to implement the steps because they draw heavily on your technical-solution design experience. Hence, we must begin by studying the design of relatively simple programs like the one described in this section.

The evaluated and tested code is as follows:

```
%
% The projectile problem with zero air resistance
% in a gravitational field with constant g.
%
% Written by D. T. Valentine ........ September 2006
     % Revised by D. T. Valentine ........  November 2008
% An eight-step structure plan applied in MATLAB:
%
% 1. Definition of the input variables.
%
g = 9.81; % Gravity in m/s/s.
vo  = input('What is the launch speed in m/s? ');
tho = input('What is the launch angle in degrees? ');
tho = pi*tho/180; % Conversion of degrees to radians.
%
% 2. Calculate the range and duration of the flight.
%
txmax = (2*vo/g) * sin(tho);
xmax = txmax * vo * cos(tho);
%
% 3. Calculate the sequence of time steps to compute trajectory.
%
dt = txmax/100;
t = 0:dt:txmax;
%
```

```
% 4. Compute the trajectory.
%
x = (vo * cos(tho)) .* t;
y = (vo * sin(tho)) .* t -(g/2) .* t.^2;
%
% 5. Compute the speed and angular direction of the projectile.
% Note that vx = dx/dt, vy = dy/dt.
%
vx = vo * cos(tho);
vy = vo * sin(tho) -g .* t;
v = sqrt(vx.*vx + vy.*vy);
        th = (180/pi) .* atan2(vy,vx);
%
% 6. Compute the time, horizontal distance at maximum altitude.
%
tymax = (vo/g) * sin(tho);
        xymax = xmax/2;
ymax = (vo/2) * tymax * sin(tho);
%
% 7. Display ouput.
%
disp([' Range in m = ',num2str(xmax), ...
             ' Duration in s = ', num2str(txmax)])
        disp(' ')
disp([' Maximum altitude in m = ',num2str(ymax), ...
             ' Arrival in s = ', num2str(tymax)])
plot(x,y,'k',xmax,y(size(t)),'o',xmax/2,ymax,'o')
title([' Projectile flight path, vo =',num2str(vo), ...
                ' th =', num2str(180*th/pi)])
xlabel(' x '), ylabel(' y ')      % Plot of Figure 1.
figure % Creates a new figure.
plot(v,th,'r')
title(' Projectile speed vs. angle ')
xlabel(' V '), ylabel(' \theta ')   % Plot of Figure 2.
%
% 8. Stop.
%
```

*Steps 6 and 7.* The program was evaluated by executing a number of values of the launch angle and launch speed within the required specifications. The angle of 45 degrees was checked to determine that the maximum range occurred at this angle for all specified launch speeds. This is well known for the zero air resistance case in a constant $g$ force field. Executing this code for $V_o = 10$ m/s and $\theta_o = 45$ degrees, the trajectory and the plot of orientation versus speed in Figures 3.1 and 3.2, respectively, were produced.

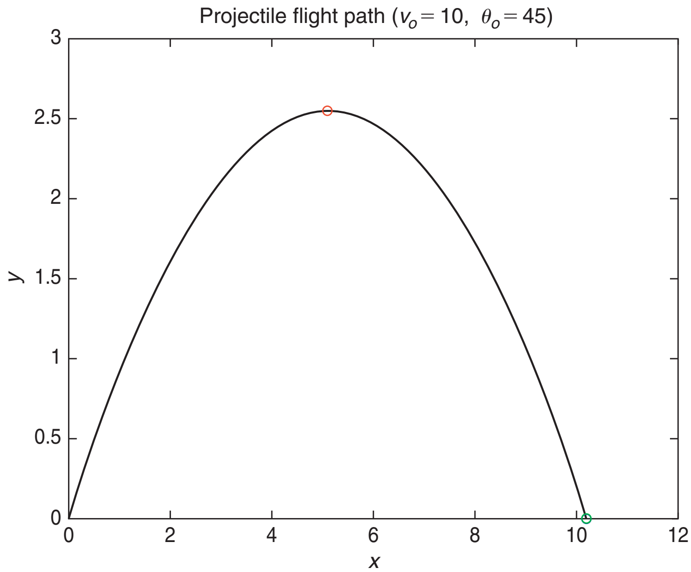


**FIGURE 3.1**

*Projectile trajectory.*

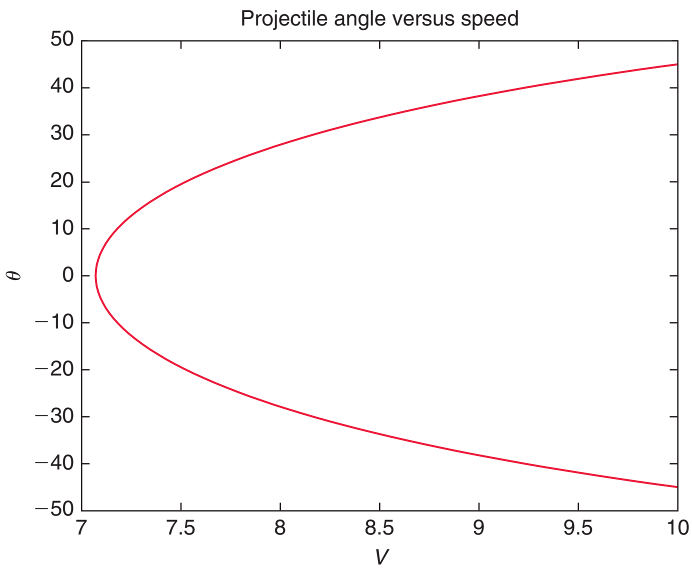


**FIGURE 3.2**

*Projectile angle versus speed.*

How can you find additional examples of MATLAB programs (good ones or otherwise) to help develop tools to solve your own problems? We all recognize that examples aren't a bad way of learning to use tools. New tools are continually being developed by the users of MATLAB. If one proves to be of more general use, MathWorks may include it in their list of products (if, of course, the tools' author desires this). There are also many examples of useful scripts that are placed on the Web for anyone interested in them. They, of course, must be evaluated carefully since it is the user's responsibility, *not* the creator's, to ensure the correctness of their results. This responsibility holds for all tools applied by the engineer and the scientist. Hence, it is very important (just as in using a laboratory apparatus) that users prove to themselves that the tool they are using is indeed valid for the problem they are trying to solve.

To illustrate how easy it is to find examples of scripts, the author typed `MATLAB examples` in one of the available search engines and found the following (among many others):

```
>> t = (0:.1:2*pi)';
>> subplot(2,2,1)
>> plot(t,sin(t))
>> subplot(2,2,2)
>> plot(t,cos(t))
>> subplot(2,2,3)
>> plot(t,exp(t))
>> subplot(2,2,4)
>> plot(t,1./(1+t.^2))
```

This script illustrates how to put four plots in a single figure window. To check that it works, type each line in the Command Window followed by **Enter**. Note the position of each graphic; location is determined by the three integers in the `subplot` function list of arguments. Search Help via the question mark (**?**) for more information on `subplot`.

## 3.2 STRUCTURE PLAN EXAMPLES

A structure plan is typically written in what is called *pseudo-code*—that is, statements in English, mathematics, and MATLAB describing in detail how to solve a problem. You don't have to be an expert in any particular computer language to understand pseudo-code. A structure plan may be written at a number of levels, each of increasing complexity, as the logical structure of the program is developed.

Suppose we want to write a script to convert a temperature on the Fahrenheit scale (where water freezes and boils at 32° and 212°, respectively) to the

Celsius scale. A first-level structure plan might be a simple statement of the problem:

1. Initialize Fahrenheit temperature
2. Calculate and display Celsius temperature
3. Stop

Step 1 is pretty straightforward. Step 2 needs elaborating, so the second-level plan could be something like this:

1. Initialize Fahrenheit temperature ($F$)
2. Calculate Celsius temperature ($C$) as follows:
   2.1. Subtract 32 from $F$ and multiply by 5/9
3. Display the value of $C$
4. Stop

There are no hard and fast rules for writing structure plans. The essential point is to cultivate the mental discipline of getting the problem logic clear before attempting to write the program. The top-down approach of structure plans means that the overall structure of a program is clearly thought out before you have to worry about the details of syntax (coding). This reduces the number of errors enormously.

A script to implement this is as follows:

```
% Script file to convert temperatures from F to C
% D.T.V. ......... October 2006/November 2008
%
F = input(' Temperature in degrees F: ')
      C = (F-32)*5/9;
disp([' Temperature in degrees C = ',num2str(C)])
% STOP
```

Two checks of the tool were done. They were for $F = 32$, which gave $C = 0$, and $F = 212$, which gave $C = 100$. The results were found to be correct and hence this simple script is, as such, validated.

### 3.2.1 Quadratic equation

When you were at school you probably solved hundreds of quadratic equations of the form

$$ax^2 + bx + c = 0$$

A structure plan of the *complete* algorithm for finding the solution(s) $x$, given any values of $a$, $b$, and $c$, is shown in Figure 3.3. Figure 3.4 shows the graph of a quadratic equation with real unequal roots.

1. Start
2. Input data ($a$, $b$, $c$)
3. If $a = 0$ then
    If $b = 0$ then
      If $c = 0$ then
        Display 'Solution indeterminate'
      else
        Display 'There is no solution'
    else
      $x = -c/b$
      Display $x$ (only one root: equation is linear)
  else if $b^2 < 4ac$ then
    Display 'Complex roots'
  else if $b^2 = 4ac$ then
    $x = -b/(2a)$
    Display $x$ (equal roots)
  else
    $x_1 = (-b + \sqrt{b^2 - 4ac})/(2a)$
    $x_2 = (-b - \sqrt{b^2 - 4ac})/(2a)$
    Display $x_1$, $x_2$
4. Stop

**FIGURE 3.3**

*Quadratic equation structure plan.*

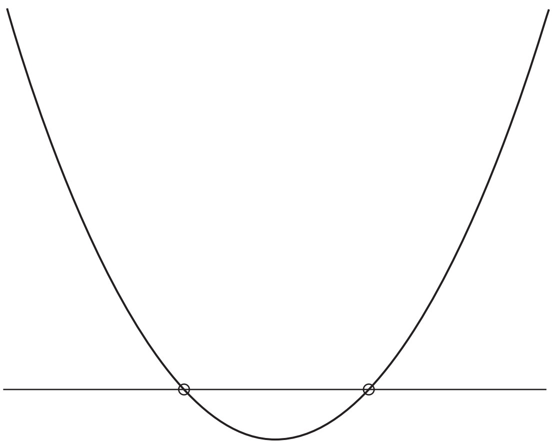

**FIGURE 3.4**

*Graph of a quadratic equation with real unequal roots indicated by* o.

When you write the program, you can type the structure plan into the MATLAB Editor, as described in Chapter 2. Use cut and paste to make another copy of the plan below the first one, and translate the second copy into MATLAB statements. If it is a good structure plan, it should translate line for line. Then comment out the original structure plan with **Text** → **Comment**, so you can save it with the program.

## 3.3 STRUCTURED PROGRAMMING WITH FUNCTIONS

Many examples later in this book will be rather involved. More advanced programs like these should be *structured* by means of your own *function M-files*. These are dealt with in detail in Chapter 10. A function M-file is a script file (i.e., a file with an `.m` extension) that you can "call" interactively, or from other scripts, in specific ways.

The "main" script will look very much like a first-level structure plan of the problem. For example, the quadratic equation problem may be structure-planned at the first level as follows:

1. Input the data
2. Find and display the solution(s)
3. Stop

Using a function file that you created and called `quadratic.m` to do the dirty work could be translated directly into the following MATLAB script:

```
a = input( 'Enter coefficients: ' );
x = quadratic(a);
```

(The details on coding this particular problem are left as an exercise in Chapter 10.)

## SUMMARY

- An algorithm is a systematic logical procedure for solving a problem.
- A structure plan is a representation of an algorithm in pseudo-code.
- A function M-file is a script file designed to handle a particular task that may be activated (invoked) whenever needed.

## CHAPTER EXERCISES

The problems in these exercises should all be structure-planned before being written up as MATLAB programs (where appropriate).

**3.1.** The structure plan in this example defines a geometric construction. Carry out the plan by sketching the construction:

1. Draw two perpendicular *x*- and *y*-axes
2. Draw the points *A* (10, 0) and *B* (0, 1)
3. While *A* does not coincide with the origin repeat:
    Draw a straight line joining *A* and *B*

Move $A$ one unit to the left along the $x$-axis
Move $B$ one unit up on the $y$-axis

4. Stop

**3.2.** Consider the following structure plan, where $M$ and $N$ represent MATLAB variables:

1. Set $M = 44$ and $N = 28$
2. While $M$ not equal to $N$ repeat:
   While $M > N$ repeat:
      Replace value of $M$ by $M - N$
   While $N > M$ repeat:
      Replace value of $N$ by $N - M$
3. Display $M$
4. Stop

**(a)** Work through the structure plan, sketching the contents of $M$ and $N$ during execution. Give the output.

**(b)** Repeat (a) for $M = 14$ and $N = 24$.

**(c)** What general arithmetic procedure does the algorithm carry out (try more values of $M$ and $N$ if necessary)?

**3.3.** Write a program to convert a Fahrenheit temperature to Celsius. Test it on the data in Exercise 2.11 (where the reverse conversion is done).

**3.4.** Write a script that inputs any two numbers (which may be equal) and displays the larger one with a suitable message or, if they are equal, displays a message to that effect.

**3.5.** Write a script for the general solution to the quadratic equation $ax^2 + bx + c = 0$. Use the structure plan in Figure 3.3. Your script should be able to handle all possible values of the data $a$, $b$, and $c$. Try it out on the following values:

**(a)** 1, 1, 1 (complex roots)

**(b)** 2, 4, 2 (equal roots of $-1.0$)

**(c)** 2, 2, $-12$ (roots of 2.0 and $-3.0$)

The structure plan in Figure 3.3 is for programming languages that cannot handle complex numbers. MATLAB can. Adjust your script so that it can also find complex roots. Test it on case (a); the roots are $-0.5 \pm 0.866i$.

**3.6.** Develop a structure plan for the solution to two simultaneous linear equations (i.e., the equations of two straight lines). Your algorithm must be able to handle all possible situations; that is, lines intersecting, parallel, or coincident. Write a program to implement your algorithm, and test it on some equations for which you know the solutions, such as

$$x + y = 3$$

$$2x - y = 3$$

($x = 2$, $y = 1$). *Hint*: Begin by deriving an algebraic formula for the solution to the system:

$$ax + by = c$$

$$dx + ey = f$$

The program should input the coefficients $a$, $b$, $c$, $d$, $e$, and $f$.

(*Continued*)

We will see in Chapter 16 that MATLAB has a very elegant way of solving systems of equations directly, using matrix algebra. However, it is good for the development of your programming skills to do it the long way, as in this exercise.

**3.7.** We wish to examine the motion of a damped harmonic oscillator. The small amplitude oscillation of a unit mass attached to a spring is given by the formula $y = e^{-(R/2)t} \sin(\omega_1 t)$, where $\omega_1^2 = \omega_o^2 - R^2/4$ is the square of the natural frequency of the oscillation with damping (i.e., with resistance to motion); $\omega_o^2 = k$ is the square of the natural frequency of undamped oscillation; $k$ is the spring constant; and $R$ is the damping coefficient. Consider $k = 1$ and vary $R$ from 0 to 2 in increments of 0.5. Plot $y$ versus $t$ for $t$ from 0 to 10 in increments of 0.1. *Hint*: Develop a solution procedure by working backwards through the problem statement. Starting at the end of the problem statement, the solution procedure requires the programmer to assign the input variables first followed by the execution of the formula for the amplitude and ending with the output in graphical form.

**3.8.** Let's examine the shape of a uniform cable hanging under its own weight. The shape is described by the formula $y = \cosh(x/c)$. This shape is called a uniform catenary. The parameter $c$ is the vertical distance from $y = 0$ where the bottom of the catenary is located. Plot the shape of the catenary between $x = -10$ and $x = 10$ for $c = 5$. Compare this with the same result for $c = 4$. *Hint*: The hyperbolic cosine, `cosh`, is a built-in MATLAB function that is used in a similar way to the sine function, `sin`.