Chapter 7
User-Defined Functions and Function Files

A simple function in mathematics, $f(x)$, associates a unique number to each value of x. The function can be expressed in the form $y = f(x)$, where $f(x)$ is usually a mathematical expression in terms of x. A value of y (output) is obtained when a value of x (input) is substituted in the expression. Many functions are programmed inside MATLAB as built-in functions, and can be used in mathematical expressions simply by typing their name with an argument (see Section 1.5); examples are `sin(x)`, `cos(x)`, `sqrt(x)`, and `exp(x)`. Frequently, in computer programs, there is a need to calculate the value of functions that are not built in. When a function expression is simple and needs to be calculated only once, it can be typed as part of the program. However, when a function needs to be evaluated many times for different values of arguments, it is convenient to create a "user-defined" function. Once a user-defined function is created (saved) it can be used just like the built-in functions.

A user-defined function is a MATLAB program that is created by the user, saved as a function file, and then used like a built-in function. The function can be a simple single mathematical expression or a complicated and involved series of calculations. In many cases it is actually a subprogram within a computer program. The main feature of a function file is that it has an input and an output. This means that the calculations in the function file are carried out using the input data, and the results of the calculations are transferred out of the function file by the output. The input and the output can be one or several variables, and each can be a scalar, a vector, or an array of any size. Schematically, a function file can be illustrated by:

Input data
Function File
Output data

A very simple example of a user-defined function is a function that calculates the maximum height that a ball reaches when thrown upward with a certain velocity. For a velocity v_0, the maximum height h_{max} is given by $h_{\max} = \frac{v_0^2}{2g}$, where g is the gravitational acceleration. In function form this can be written as $h_{\max}(v_0) = \frac{v_0^2}{2g}$. In this case the input to the function is the velocity (a number), and the output is the maximum height (a number). For example, in SI units (g = 9.81 m/s^2) if the input is 15 m/s, the output is 11.47 m.

15 m/s
Function File
11.47 m

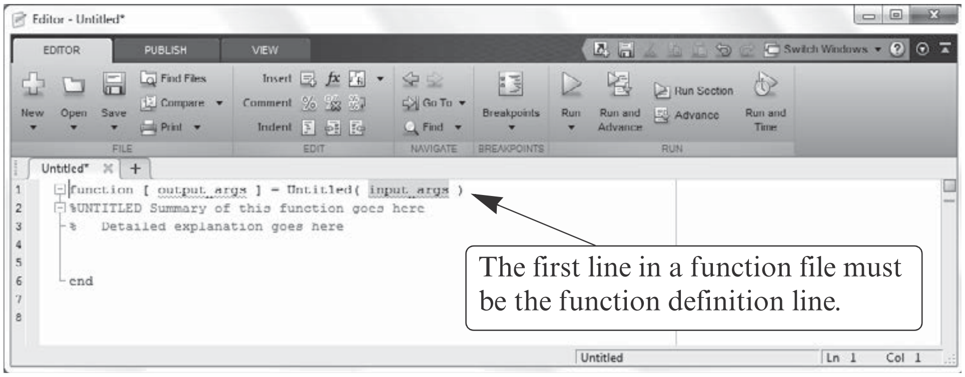
In addition to being used as math functions, user-defined functions can be used as subprograms in large programs. In this way large computer programs can be made up of smaller "building blocks" that can be tested independently. Function files are similar to subroutines in Basic and Fortran, procedures in Pascal, and functions in C.

The fundamentals of user-defined functions are explained in Sections 7.1 through 7.7. In addition to user-defined functions that are saved in separate function files and called for use in a computer program, MATLAB provides an option to define and use a user-defined math function within a computer program (not in a separate file). This can be done by using anonymous function, which is presented in Section 7.8. There are built-in and user-defined functions that have to be supplied with other functions when they are called. These functions, which in MATLAB are called function functions, are introduced in Section 7.9. The last two sections cover subfunctions and nested functions. Both are methods for incorporating two or more user-defined functions in a single function file.

7.1 Creating a Function File

Function files are created and edited, like script files, in the Editor/Debugger Window. This window is opened from the Command Window. In the Toolstrip select **New**, then select **Function**. Once the Editor/Debugger Window opens, it looks like that shown in Figure 7-1. The editor contains several pre-typed lines that outline the structure of a function file. The first line is the function definition line, which is followed by comments the describe the function. Next comes the program (the empty lines 4 and 5 in Figure 7-1), and the last line is an `end` statement, which is optional. The structure of a function file is described in detail in the next section.

Note: The Editor/Debugger Window can also be opened (as was described in Chapter 1) by clicking on the **New Script** icon in the Toolstrip, or by clicking **New** in the Toolstrip and then selecting **Script** from the menu that open. The



```
function [ output_args ] = Untitled( input_args )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here


end
```

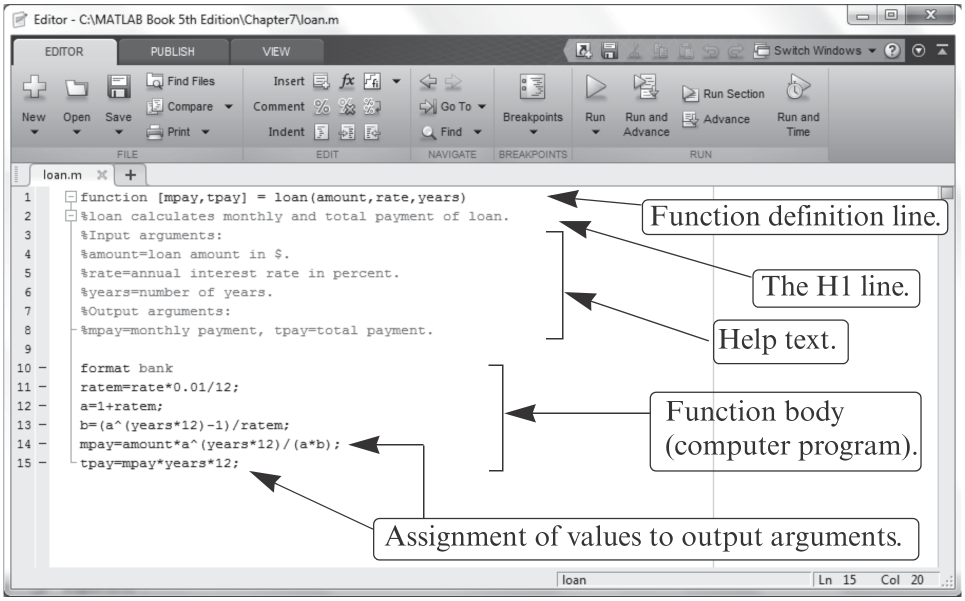
Editor - Untitled*
The first line in a function file must be the function definition line.

Figure 7-1: The Editor/Debugger Window.

window that opens is empty, without any pre-typed lines. In general, the Editor/ Debugger Window can be used for writing a script file or a function file.

7.2 *Structure of a Function File*

The structure of a typical complete function file is shown in Figure 7-2. This particular function calculates the monthly payment and the total payment of a loan. The inputs to the function are the amount of the loan, the annual interest rate, and the duration of the loan (number of years). The output from the function is the monthly payment and the total payment.



```
function [mpay,tpay] = loan(amount,rate,years)
%loan calculates monthly and total payment of loan.
%Input arguments:
%amount=loan amount in $.
%rate=annual interest rate in percent.
%years=number of years.
%Output arguments:
%mpay=monthly payment, tpay=total payment.

format bank
ratem=rate*0.01/12;
a=1+ratem;
b=(a^(years*12)-1)/ratem;
mpay=amount*a^(years*12)/(a*b);
tpay=mpay*years*12;
```

Editor - C:\MATLAB Book 5th Edition\Chapter7\loan.m
Function definition line.
The H1 line.
Help text.
Function body (computer program).
Assignment of values to output arguments.

Figure 7-2: Structure of a typical function file.

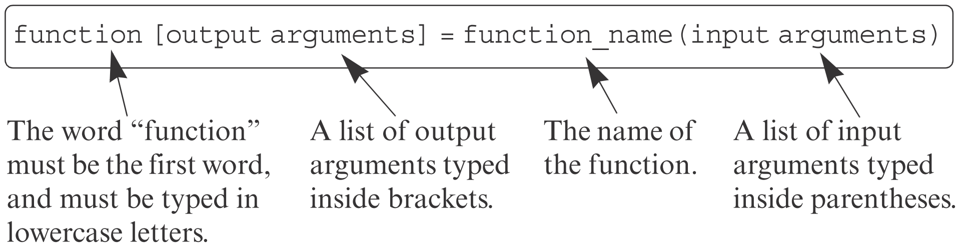
The various parts of the function file are described in detail in the following sections.

7.2.1 Function Definition Line

The first executable line in a function file *must* be the function definition line. Otherwise the file is considered a script file. The function definition line:

- Defines the file as a function file
- Defines the name of the function
- Defines the number and order of the input and output arguments

The form of the function definition line is:



```
function [output arguments] = function_name(input arguments)
```

The word "function" must be the first word, and must be typed in lowercase letters.
A list of output arguments typed inside brackets.
The name of the function.
A list of input arguments typed inside parentheses.

The word "function," typed in lowercase letters, must be the first word in the function definition line. On the screen the word function appears in blue. The function name is typed following the equal sign. The name can be made up of letters, digits, and the underscore character (the name cannot include a space). The rules for the name are the same as the rules for naming variables described in Section 1.6.2. It is good practice to avoid names of built-in functions and names of variables already defined by the user or predefined by MATLAB.

7.2.2 Input and Output Arguments

The input and output arguments are used to transfer data into and out of the function. The input arguments are listed inside parentheses following the function name. Usually, there is at least one input argument, although it is possible to have a function that has no input arguments. If there are more than one, the input arguments are separated with commas. The computer code that performs the calculations within the function file is written in terms of the input arguments and assumes that the arguments have assigned numerical values. This means that the mathematical expressions in the function file must be written according to the dimensions of the arguments, since the arguments can be scalars, vectors, or arrays. In the example shown in Figure 7-2 there are three input arguments (`amount,rate,years`), and in the mathematical expressions they are assumed to be scalars. The actual values of the input arguments are assigned when the function is used (called). Similarly, if the input arguments are vectors or arrays, the mathematical expressions in the function body must be written to follow linear algebra or element-by-element calculations.

The output arguments, which are listed inside brackets on the left side of the assignment operator in the function definition line, transfer the output from the function file. Function files can have zero, one, or several output arguments. If there are more than one, the output arguments are separated with commas. If there is only one output argument, it can be typed without brackets. *For the function file to work, the output arguments must be assigned values in the computer program that is in the function body.* In the example in Figure 7-2 there are two output arguments, `mpay` and `tpay`. When a function does not have an output argument, the assignment operator in the function definition line can be omitted. A function without an output argument can, for example, generate a plot or write data to a file.

It is also possible to transfer strings into a function file. This is done by typing the string as part of the input variables (text enclosed in single quotes). Strings can be used to transfer names of other functions into the function file.

Usually, all the input to, and the output from, a function file transferred through the input and output arguments. In addition, however, all the input and output features of script files are valid and can be used in function files. This means that any variable that is assigned a value in the code of the function file will be displayed on the screen unless a semicolon is typed at the end of the command. In addition, the `input` command can be used to input data interactively, and the `disp`, `fprintf`, and `plot` commands can be used to display information on the screen, save to a file, or plot figures just as in a script file. The following are examples of function definition lines with different combinations of input and output arguments.

Function definition line	**Comments**
function[mpay,tpay]= loan(amount,rate,years)	Three input arguments, two output arguments.
function [A] = RectArea(a,b)	Two input arguments, one output argument.
function A = RectArea(a,b)	Same as above; one output argument can be typed without the brackets.
function [V, S] = SphereVolArea(r)	One input variable, two output variables.
function trajectory(v,h,g)	Three input arguments, no output arguments.

7.2.3 The H1 Line and Help Text Lines

The H1 line and help text lines are comment lines (lines that begin with the percent, %, sign) following the function definition line. They are optional but are frequently used to provide information about the function. The H1 line is the first comment line and usually contains the name and a short definition of the function. When a user types (in the Command Window) `lookfor a_word`, MATLAB searches for `a_word` in the H1 lines of all the functions, and if a match is found, the H1 line that contains the match is displayed.

The help text lines are comment lines that follow the H1 line. These lines contain an explanation of the function and any instructions related to the input and output arguments. The comment lines that are typed between the function definition line and the first non-comment line (the H1 line and the help text) are displayed when the user types `help function_name` in the Command Window. This is true for MATLAB built-in functions as well as the user-defined functions. For example, for the function `loan` in Figure 7-2, if `help loan` is typed in the Command Window (make sure the current directory or the search path includes the directory where the file is saved), the display on the screen is:

```
>> help loan
loan calculates monthly and total payment of loan.
Input arguments:
amount=loan amount in $.
rate=annual interest rate in percent.
years=number of years.
Output arguments:
mpay=monthly payment, tpay=total payment.
```

A function file can include additional comment lines in the function body. These lines are ignored by the `help` command.

7.2.4 Function Body

The function body contains the computer program (code) that actually performs the computations. The code can use all MATLAB programming features. This includes calculations, assignments, any built-in or user-defined functions, flow control (conditional statements and loops) as explained in Chapter 6, comments, blank lines, and interactive input and output.

7.3 LOCAL AND GLOBAL VARIABLES

All the variables in a function file are local (the input and output arguments and any variables that are assigned values within the function file). This means that the variables are defined and recognized only inside the function file. When a function file is executed, MATLAB uses an area of memory that is separate from the workspace (the memory space of the Command Window and the script files). In a function file the input variables are assigned values each time the function is called. These variables are then used in the calculations within the function file. When the function file finishes its execution, the values of the output arguments are transferred to the variables that were used when the function was called. All this means that a function file can have variables with the same names as variables in the Command Window or in script files. The function file does not recognize variables with the same names as have been assigned values outside the function. The assignment of values to these variables in the function file will not change their assignment elsewhere.

Each function file has its own local variables, which are not shared with other functions or with the workspace of the Command Window and the script files. It is possible, however, to make a variable common (recognized) in several different function files, and perhaps in the workspace too. This is done by declaring the variable global with the `global` command, which has the form:

```
global variable_name
```

Several variables can be declared global by listing them, separated with spaces, in the global command. For example:

```
global GRAVITY_CONST FrictionCoefficient
```

- The variable has to be declared global in every function file that the user wants it to be recognized in. The variable is then common only to these files.
- The `global` command must appear before the variable is used. It is recommended to enter the `global` command at the top of the file.
- The `global` command has to be entered in the Command Window, or in a script file, for the variable to be recognized in the workspace.
- The variable can be assigned, or reassigned, a value in any of the locations in which it is declared common.
- The use of long descriptive names (or all capital letters) is recommended for global variables in order to distinguish them from regular variables.

7.4 Saving a Function File

A function file must be saved before it can be used. This is done, as with a script file, by choosing **Save as . . .** from the **File** menu, selecting a location (many students save to a flash drive), and entering the file name. It is highly recommended that the file be saved with a name that is identical to the function name in the function definition line. In this way the function is called (used) by using the function name. (If a function file is saved with a different name, the name it is saved under must be used when the function is called.) Function files are saved with the extension .m. Examples:

Function definition line	**File name**
function [mpay,tpay] = loan(amount,rate,years)	loan.m
function [A] = RectArea(a,b)	RectArea.m
function [V, S] = SphereVolArea(r)	SphereVolArea.m
function trajectory(v,h,g)	trajectory.m

7.5 Using a User-Defined Function

A user-defined function is used in the same way as a built-in function. The function can be called from the Command Window, from a script file, or from another function. To use the function file, the folder where it is saved must either be in the current folder or be in the search path (see Sections 1.8.3 and 1.8.4).

A function can be used by assigning its output to a variable (or variables), as a part of a mathematical expression, as an argument in another function, or just by typing its name in the Command Window or in a script file. In all cases the user must know exactly what the input and output arguments are. An input argument can be a number, a computable expression, or a variable that has an assigned value. The arguments are assigned according to their position in the input and output argument lists in the function definition line.

Two of the ways that a function can be used are illustrated below with the user-defined `loan` function in Figure 7-2, which calculates the monthly and total payments (two output arguments) of a loan. The input arguments are the loan amount, annual interest rate, and the length (number of years) of the loan. In the first illustration the `loan` function is used with numbers as input arguments:

```
>> [month total]=loan(25000,7.5,4)

month =
        600.72
total =
      28834.47
```

First argument is loan amount, second is interest rate, and third is number of years.

In the second illustration the `loan` function is used with two pre-assigned variables and a number as the input arguments:

```
>> a=70000;  b=6.5;
>> [x y]=loan(a,b,30)
```

Define variables `a` and `b`.

Use `a`, `b`, and the number 30 for input arguments and `x` (monthly pay) and `y` (total pay) for output arguments.

```
x =
        440.06
y =
     158423.02
```

7.6 EXAMPLES OF SIMPLE USER-DEFINED FUNCTIONS

Sample Problem 7-1: User-defined function for a math function

Write a function file (name it `chp7one`) for the function $f(x) = \dfrac{x^4\sqrt{3x+5}}{(x^2+1)^2}$. The input to the function is x and the output is $f(x)$. Write the function such that x can be a vector. Use the function to calculate:

(a) $f(x)$ for $x = 6$.
(b) $f(x)$ for $x = 1, 3, 5, 7, 9$, and 11.

Solution

The function file for the function $f(x)$ is:

```
function y=chp7one(x)                        Function definition line.
y=(x.^4.*sqrt(3*x+5))./(x.^2+1).^2;  Assignment to output argument.
```

Note that the mathematical expression in the function file is written for element-by-element calculations. In this way if x is a vector, y will also be a vector. The function is saved and then the search path is modified to include the directory where the file was saved. As shown below, the function is used in the Command Window.

(a) Calculating the function for $x = 6$ can be done by typing `chp7one(6)` in the Command Window, or by assigning the value of the function to a new variable:

```
>> chp7one(6)
ans =
    4.5401
>> F=chp7one(6)
F =
    4.5401
```

(b) To calculate the function for several values of x, a vector with the values of x is created and then used for the argument of the function.

```
>> x=1:2:11
x =
     1     3     5     7     9    11
```

```
>> chp7one(x)
ans =
    0.7071    3.0307    4.1347    4.8971    5.5197    6.0638
```

Another way is to type the vector x directly in the argument of the function.

```
>> H=chp7one([1:2:11])
H =
    0.7071    3.0307    4.1347    4.8971    5.5197    6.0638
```

Sample Problem 7-2: Converting temperature units

Write a user-defined function (name it `FtoC`) that converts temperature in degrees F to temperature in degrees C. Use the function to solve the following problem. The change in the length of an object, ΔL, due to a change in the temperature, ΔT, is given by: $\Delta L = \alpha L \Delta T$, where α is the coefficient of thermal expansion. Determine the change in the area of a rectangular (4.5 m by 2.25 m) aluminum ($\alpha = 23 \cdot 10^{-6}$ 1/°C) plate if the temperature changes from 40°F to 92°F.

Solution

A user-defined function that converts degrees F to degrees C is:

```
function C=FtoC(F)                          Function definition line.
%FtoC converts degrees F to degrees C
C=5*(F-32)./9;                       Assignment to output argument.
```

A script file (named Chapter7Example2) that calculates the change of the area of the plate due to the temperature is:

```
a1=4.5; b1=2.25; T1=40; T2=92; alpha=23e-6;
deltaT=FtoC(T2)-FtoC(T1);   Using the FtoC function to calculate the
                            temperature difference in degrees C.
a2=a1+alpha*a1*deltaT;               Calculating the new length.
b2=b1+alpha*b1*deltaT;               Calculating the new width.
AreaChange=a2*b2-a1*b1;      Calculating the change in the area.
fprintf('The  change  in  the  area  is  %6.5f  meters
square.',AreaChange)
```

Executing the script file in the Command Window gives the solution:

```
>> Chapter7Example2
The change in the area is 0.01346 meters square.
```

7.7 *Comparison between Script Files and Function Files*

Students who are studying MATLAB for the first time sometimes have difficulty understanding exactly the differences between script and function files, since for many of the problems that they are asked to solve using MATLAB, either type of file can be used. The similarities and differences between script and function files are summarized below.

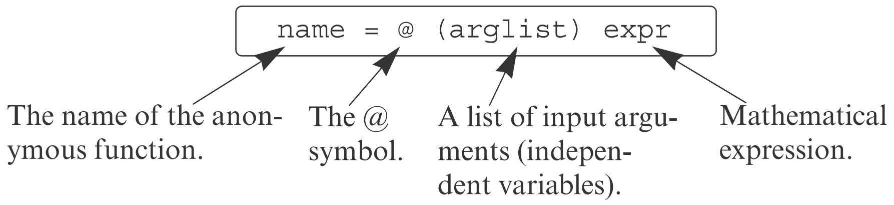
- Both script and function files are saved with the extension .m (that is why they are sometimes called M-files).
- The first executable line in a function file is (must be) the function definition line.
- The variables in a function file are local. The variables in a script file are recognized in the Command Window.
- Script files can use variables that have been defined in the workspace.
- Script files contain a sequence of MATLAB commands (statements).
- Function files can accept data through input arguments and can return data through output arguments.
- When a function file is saved, the name of the file should be the same as the name of the function.
- A user-defined function is used in the same way as a built-in function. It can be used (called) in the Command Window, in a script file, or in another function.

7.8 *Anonymous Functions*

User-defined functions written in function files can be used for simple mathematical functions, for large and complicated math functions that require extensive programming, and as subprograms in large computer programs. In cases when the value of a relatively simple mathematical expression has to be calculated many times within a program, MATLAB provides the option of using anonymous functions. An anonymous function is a user-defined function that is defined and written within the computer code (not in a separate function file) and is then used in the code. Anonymous functions can be defined in any part of MATLAB (in the Command Window, in script files, and inside regular user-defined functions).

An anonymous function is a simple (one-line) user-defined function that is defined without creating a separate function file (m-file). Anonymous functions can be constructed in the Command Window, within a script file, or inside a regular user-defined function.

An anonymous function is created by typing the following command:


name = @ (arglist) expr
The name of the anonymous function.
The @ symbol.
A list of input arguments (independent variables).
Mathematical expression.

A simple example is `cube = @ (x) x^3`, which calculates the cube of the input argument.

- The command creates the anonymous function and assigns a handle for the function to the variable name on the left-hand side of the = sign. (Function handles provide means for using the function and passing it to other functions; see Section 7.9.1.)
- The `expr` consists of a single valid mathematical MATLAB expression.
- The mathematical expression can have one or several independent variables. The independent variable(s) is (are) entered in the `(arglist)`. Multiple independent variables are separated with commas. An example of an anonymous function that has two independent variables is: `circle = @ (x,y) 16*x^2+9*y^2`.
- The mathematical expression can include any built-in or user-defined functions.
- The expression must be written according to the dimensions of the arguments (element-by-element or linear algebra calculations).
- The expression can include variables that are already defined when the anonymous function is defined. For example, if three variables `a`, `b`, and `c` are defined (have assigned numerical values), then they can be used in the expression of the anonymous function `parabola = @ (x) a*x^2+b*x+c`.

Important note: MATLAB captures the values of the predefined variables when the anonymous function is defined. This means that if new values are subsequently assigned to the predefined variables, the anonymous function is not changed. The anonymous function has to be redefined in order for the new values of the predefined variables to be used in the expression.

Using an anonymous function:

- Once an anonymous function is defined, it can be used by typing its name and a value for the argument (or arguments) in parentheses (see examples that follow).
- Anonymous functions can also be used as arguments in other functions (see Section 7.9.1).

Example of an anonymous function with one independent variable:

The function $f(x) = \frac{e^{x^2}}{\sqrt{x^2+5}}$ can be defined (in the Command Window) as an anonymous function for x as a scalar by:

```
>> FA = @ (x) exp(x^2)/sqrt(x^2+5)
FA =
    @(x)exp(x^2)/sqrt(x^2+5)
```

If a semicolon is not typed at the end, MATLAB responds by displaying the function. The function can then be used for different values of x, as shown below.

```
>> FA(2)
ans =
   18.1994
>> z = FA(3)
z =
  2.1656e+003
```

If x is expected to be an array, with the function calculated for each element, then the function must be modified for element-by-element calculations.

```
>> FA = @ (x) exp(x.^2)./sqrt(x.^2+5)
FA =
    @(x)exp(x.^2)./sqrt(x.^2+5)
>> FA([1 0.5 2])
ans =
    1.1097    0.5604   18.1994
```

Using a vector as input argument.

Example of an anonymous function with several independent variables:

The function $f(x, y) = x^2 - 4xy + y^2$ can be defined as an anonymous function by:

```
>> HA = @ (x,y) 2*x^2 - 4*x*y + y^2
HA =
    @(x,y)2*x^2-4*x*y+y^2
```

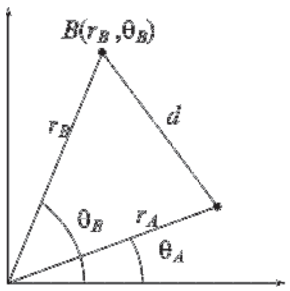
Then the anonymous function can be used for different values of x and y. For example, typing `HA(2,3)` gives:

```
>> HA(2,3)
ans =
    -7
```

Another example of using an anonymous function with several arguments is shown in Sample Problem 6-3.

Sample Problem 7-3: Distance between points in polar coordinates

Write an anonymous function that calculates the distance between two points in a plane when the position of the points is given in polar coordinates. Use the anonymous function to calculate the distance between point A (2, π/6) and point B (5, 3π/4).



Solution

The distance between two points in polar coordinates can be calculated by using the Law of Cosines:

$$d = \sqrt{r_A^2 + r_B^2 - 2r_A r_B \cos(\theta_A - \theta_B)}$$

The formula for the distance is entered as an anonymous function with four input arguments $(r_A, \theta_A, r_B, \theta_B)$. Then the function is used for calculating the distance between points A and B.

```
>> d= @ (rA,thetA,rB,thetB) sqrt(rA^2+rB^2-2*rA*rB*cos(thetB-thetA))
```

List of input arguments.

```
d =
  @(rA,thetA,rB,thetB)sqrt(rA^2+rB^2-2*rA*rB*cos(thetB-thetA))
>> DistAtoB = d(2,pi/6,5,3*pi/4)
DistAtoB =
    5.8461
```

The arguments are typed in the order defined in the function.

7.9 Function Functions

There are many situations where a function (Function A) works on (uses) another function (Function B). This means that when Function A is executed, it has to be provided with Function B. A function that accepts another function is called in MATLAB a function function. For example, MATLAB has a built-in function called `fzero` (Function A) that finds the zero of a math function $f(x)$ (Function B) — i.e., the value of x where $f(x) = 0$. The program in the function `fzero` is written such that it can find the zero of any $f(x)$. When `fzero` is called, the specific function to be solved is passed into `fzero`, which finds the zero of the $f(x)$. (The function `fzero` is described in detail in Chapter 9.)

A function function, which accepts another function (imported function), includes in its input arguments a name that represents the imported function. The imported function name is used for the operations in the program (code) of the function function. When the function function is used (called), the specific function that is imported is listed in its input argument. In this way different functions can be imported (passed) into the function function. There are two methods for listing the name of an imported function in the argument list of a function function. One is by using a function handle (Section 7.9.1), and the other is by typing the name of the function that is being passed in as a string expression (Section 7.9.2). The method that is used affects the way that the operations in the function function are written (this is explained in more detail in the next two sections). Using function handles is easier and more efficient, and should be the preferred method.

7.9.1 Using Function Handles for Passing a Function into a Function Function

Function handles are used for passing (importing) user-defined functions, built-in functions, and anonymous functions into function functions that can accept them. This section first explains what a function handle is, then shows how to write a user-defined function function that accepts function handles, and finally shows how to use function handles for passing functions into function functions.

Function handle:

A function handle is a MATLAB value that is associated with a function. It is a MATLAB data type and can be passed as an argument into another function. Once passed, the function handle provides means for calling (using) the function it is associated with. Function handles can be used with any kind of MATLAB function. This includes built-in functions, user-defined functions (written in function files), and anonymous functions.

- For built-in and user-defined functions, a function handle is created by typing the symbol @ in front of the function name. For example, `@cos` is the function handle of the built-in function `cos`, and `@FtoC` is the function handle of the user-defined function `FtoC` that was created in Sample Problem 7-2.
- The function handle can also be assigned to a variable name. For example, `cosHandle=@cos` assigns the handle `@cos` to `cosHandle`. Then the name `cosHandle` can be used for passing the handle.
- As anonymous functions (see Section 7.8.1), their name is already a function handle.

Writing a function function that accepts a function handle as an input argument:

As already mentioned, the input arguments of a function function (which accepts another function) includes a name (dummy function name) that represents the imported function. This dummy function (including a list of input arguments enclosed in parentheses) is used for the operations of the program inside the function function.

- The function that is actually being imported must be in a form consistent with the way that the dummy function is being used in the program. This means that both must have the same number and type of input and output arguments.

The following is an example of a user-defined function function, named `funplot`, that makes a plot of a function (any function $f(x)$ that is imported into it) between the points $x = a$ and $x = b$. The input arguments are `(Fun,a,b)`, where `Fun` is a dummy name that represents the imported function, and `a` and `b` are the end points of the domain. The function `funplot` also has a numerical output `xyout`, which is a 2×3 matrix with the values of x and $f(x)$ at the three points $x = a$, $x = (a + b)/2$, and $x = b$. Note that in the program, the dummy function `Fun` has one input argument `(x)` and one output argument `y`, which are both vectors.

A name for the function that is passed in.

```
function xyout=funplot(Fun,a,b)
% funplot makes a plot of the function Fun which is passed in
% when funplot is called in the domain [a, b].
% Input arguments are:
% Fun:  Function handle of the function to be plotted.
% a:  The first point of the domain.
% b:  The last point of the domain.
% Output argument is:
% xyout: The values of x and y at x=a, x=(a+b)/2, and x=b
% listed in a 3 by 2 matrix.

x=linspace(a,b,100);
y=Fun(x);
xyout(1,1)=a; xyout(2,1)=(a+b)/2; xyout(3,1)=b;
xyout(1,2)=y(1);
xyout(2,2)=Fun((a+b)/2);
xyout(3,2)=y(100);
plot(x,y)
xlabel('x'), ylabel('y')
```

Using the imported function to calculate $f(x)$ at 100 points.

Using the imported function to calculate $f(x)$ at the midpoint.

As an example, the function $f(x) = e^{-0.17x}x^3 - 2x^2 + 0.8x - 3$ over the domain $[0.5, 4]$ is passed into the user-defined function `funplot`. This is done in two ways: first by writing a user-defined function for $f(x)$, and then by writing $f(x)$ as an anonymous function.

Passing a user-defined function into a function function:

First, a user-defined function is written for $f(x)$. The function, named Fdemo, calculates $f(x)$ for a given value of x and is written using element-by-element operations.

```
function y=Fdemo(x)
y=exp(-0.17*x).*x.^3-2*x.^2+0.8*x-3;
```

Next, the function Fdemo is passed into the user-defined function function funplot, which is called in the Command Window. Note that a handle of the user-defined function Fdemo is entered (the handle is @Fdemo) for the input argument Fun in the user-defined function funplot.

```
>> ydemo=funplot(@Fdemo,0.5,4)
ydemo =
    0.5000   -2.9852
    2.2500   -3.5548
    4.0000    0.6235
```

Enter a handle of the user-defined function Fdemo.

In addition to the display of the numerical output, when the command is executed, the plot shown in Figure 7-3 is displayed in the Figure Window.

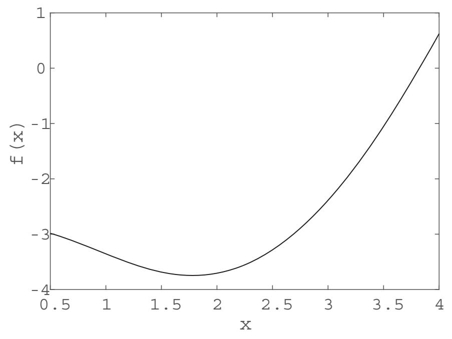


Figure 7-3: A plot of the function $f(x) = e^{-0.17x}x^3 - 2x^2 + 0.8x - 3$ **.**

Passing an anonymous function into a function function:

To use an anonymous function, the function $f(x) = e^{-0.17x}x^3 - 2x^2 + 0.8x - 3$ first has to be written as an anonymous function, and then passed into the user-defined function funplot. The following shows how both of these steps are done in the Command Window. Note that the name of the anonymous function FdemoAnony is entered without the @ sign for the input argument Fun in the user-defined function funplot (since the name is already the handle of the anonymous function).

```
>> FdemoAnony=@(x) exp(-0.17*x).*x.^3-2*x.^2+0.8*x-3
FdemoAnony =
    @(x) exp(-0.17*x).*x.^3-2*x.^2+0.8*x-3

>> ydemo=funplot(FdemoAnony,0.5,4)
ydemo =
    0.5000   -2.9852
    2.2500   -3.5548
    4.0000    0.6235
```

Create an anonymous function for $f(x)$.

Enter the name of the anonymous function (`FdemoAnony`).

In addition to the display of the numerical output in the Command Window, the plot shown in Figure 7-3 is displayed in the Figure Window.

7.9.2 Using a Function Name for Passing a Function into a Function Function

A second method for passing a function into a function function is by typing the name of the function that is being imported as a string in the input argument of the function function. The method that was used before the introduction of function handles can be used for importing user-defined functions. As mentioned, function handles are easier to use and more efficient and should be the preferred method. Importing user-defined functions by using their name is covered in the present edition of the book for the benefit of readers who need to understand programs written before MATLAB 7. New programs should use function handles.

When a user-defined function is imported by using its name, the value of the imported function inside the function function has to be calculated with the `feval` command. This is different from the case where a function handle is used, which means that there is a difference in the way that the code in the function function is written that depends on how the imported function is passed in.

The `feval` command:

The `feval` (short for "function evaluate") command evaluates the value of a function for a given value (or values) of the function's argument (or arguments). The format of the command is:

```
variable = feval('function name', argument value)
```

The value that is determined by `feval` can be assigned to a variable, or if the command is typed without an assignment, MATLAB displays `ans =` and the value of the function.

- The function name is typed as string.
- The function can be a built-in or a user-defined function.
- If there is more than one input argument, the arguments are separated with commas.

- If there is more than one output argument, the variables on the left-hand side of the assignment operator are typed inside brackets and separated with commas.

Two examples using the `feval` command with built-in functions follow.

```
>> feval('sqrt',64)
ans =
     8
>> x=feval('sin',pi/6)
x =
    0.5000
```

The following shows the use of the `feval` command with the user-defined function `loan` that was created earlier in the chapter (Figure 7-2). This function has three input arguments and two output arguments.

```
>> [M,T]=feval('loan',50000,3.9,10)      A $50,000 loan, 3.9% interest, 10 years.
M =
        502.22                            Monthly payment.
T =
      60266.47                            Total payment.
```

Writing a function function that accepts a function by typing its name as an input argument:

As already mentioned, when a user-defined function is imported by using its name, the value of the function inside the function function has to be calculated with the `feval` command. This is demonstrated in the following user-defined function function that is called `funplotS`. The function is the same as the function `funplot` from Section 7.9.1, except that the command `feval` is used for the calculations with the imported function.

```
                                           A name for the function that is passed in.
function xyout=funplotS(Fun,a,b)
% funplotS makes a plot of the function Fun which is passed
in
% when funplotS is called in the domain [a, b].
% Input arguments are:
% Fun: The function to be plotted. Its name is entered as
string expression.
% a:  The first point of the domain.
% b:  The last point of the domain.
% Output argument is:
% xyout: The values of x and y at x=a, x=(a+b)/2, and x=b
% listed in a 3 by 2 matrix.

x=linspace(a,b,100);
y=feval(Fun,x);
```

Using the imported function to calculate $f(x)$ at 100 points.

```
xyout(1,1)=a; xyout(2,1)=(a+b)/2; xyout(3,1)=b;
xyout(1,2)=y(1);
xyout(2,2)=feval(Fun,(a+b)/2);
```

Using the imported function to calculate $f(x)$ at the midpoint.

```
xyout(3,2)=y(100);
plot(x,y)
xlabel('x'), ylabel('y')
```

Passing a user-defined function into another function by using a string expression:

The following demonstrates how to pass a user-defined function into a function function by typing the name of the imported function as a string in the input argument. The function $f(x) = e^{-0.17x}x^3 - 2x^2 + 0.8x - 3$ from Section 7.9.1, created as a user-defined function named `Fdemo`, is passed into the user-defined function `funplotS`. Note that the name `Fdemo` is typed in a string for the input argument `Fun` in the user-defined function `funplotS`.

```
>> ydemoS=funplotS('Fdemo',0.5,4)
ydemoS =
    0.5000   -2.9852
    2.2500   -3.5548
    4.0000    0.6235
```

The name of the imported function is typed as a string.

In addition to the display of the numerical output in the Command Window, the plot shown in Figure 7-3 is displayed in the Figure Window.

7.10 *Subfunctions*

A function file can contain more than one user-defined function. The functions are typed one after the other. Each function begins with a function definition line. The first function is called the primary function and the rest of the functions are called subfunctions. The subfunctions can be typed in any order. The name of the function file that is saved should correspond to the name of the primary function. Each of the functions in the file can call any of the other functions in the file. Outside functions, or programs (script files), can call only the primary function. Each of the functions in the file has its own workspace, which means that in each the variables are local. In other words, the primary function and the subfunctions cannot access each other's variables (unless variables are declared to be global).

Subfunctions can help in writing user-defined functions in an organized manner. The program in the primary function can be divided into smaller tasks, each of which is carried out in a subfunction. This is demonstrated in Sample Problem 7-4.

Sample Problem 7-4: Average and standard deviation

Write a user-defined function that calculates the average and the standard deviation of a list of numbers. Use the function to calculate the average and the standard deviation of the following list of grades:
80 75 91 60 79 89 65 80 95 50 81

Solution

The average x_{ave} (mean) of a given set of n numbers $x_1, x_2, \ldots, x_n$ is given by:

$$x_{ave} = (x_1 + x_2 + \ldots + x_n) / n$$

The standard deviation is given by:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{i=n} (x_i - x_{ave})^2}{n-1}}$$

A user-defined function, named `stat`, is written for solving the problem. To demonstrate the use of subfunctions, the function file includes `stat` as a primary function, and two subfunctions called `AVG` and `StandDiv`. The function `AVG` calculates x_{ave}, and the function `StandDiv` calculates σ. The subfunctions are called by the primary function.The following listing is saved as one function file called `stat`.

```
function [me SD] = stat(v)                    The primary function.
n=length(v);
me=AVG(v,n);
SD=StandDiv(v,me,n);

function av=AVG(x,num)                        Subfunction.
av=sum(x)/num;

function Sdiv=StandDiv(x,xAve,num)            Subfunction.
xdif=x-xAve;
xdif2=xdif.^2;
Sdiv= sqrt(sum(xdif2)/(num-1));
```

The user-defined function `stat` is then used in the Command Window for calculating the average and the standard deviation of the grades:

```
>> Grades=[80 75 91 60 79 89 65 80 95 50 81];
>> [AveGrade StanDeviation] = stat(Grades)
AveGrade =
   76.8182
StanDeviation =
   13.6661
```

7.11 *Nested Functions*

A nested function is a user-defined function that is written inside another user-defined function. The portion of the code that corresponds to the nested function starts with a function definition line and ends with an end statement. An end statement must also be entered at the end of the function that contains the nested function. (Normally, a user-defined function does not require a terminating end statement. However, an end statement is required if the function contains one or more nested functions.) Nested functions can also contain nested functions. Obviously, having many levels of nested functions can be confusing. This section considers only two levels of nested functions.

One nested function:

The format of a user-defined function A (called the primary function) that contains one nested function B is:

```
function y=A(a1,a2)
.......
   function z=B(b1,b2)
   .......
   end
.......
end
```

- Note the end statements at the ends of functions B and A.
- The nested function B can access the workspace of the primary function A, and the primary function A can access the workspace of the function B. This means that a variable defined in the primary function A can be read and redefined in nested function B and vice versa.
- Function A can call function B, and function B can call function A.

Two (or more) nested functions at the same level:

The format of a user-defined function A (called the primary function) that contains two nested functions B and C at the same level is:

```
function y=A(a1,a2)
.......
   function z=B(b1,b2)
   .......
   end
.......
   function w=C(c1,c2)
   .......
   end
.......
end
```

- The three functions can access the workspace of each other.
- The three functions can call each other.

As an example, the following user-defined function (named `statNest`), with two nested functions at the same level, solves Sample Problem 7-4. Note that the nested functions are using variables (`n` and `me`) that are defined in the primary function.

```
function [me SD]=statNest(v)                    The primary function.
n=length(v);
me=AVG(v);

    function av=AVG(x)                          Nested function.
    av=sum(x)/n;
    end

    function Sdiv=StandDiv(x)                   Nested function.
    xdif=x-me;
    xdif2=xdif.^2;
    Sdiv= sqrt(sum(xdif2)/(n-1));
    end

SD=StandDiv(v);
end
```

Using the user-defined function `statNest` in the Command Window for calculating the average of the grade data gives:

```
>> Grades=[80 75 91 60 79 89 65 80 95 50 81];
>> [AveGrade StanDeviation] = statNest(Grades)
```

```
AveGrade =
   76.8182
StanDeviation =
   13.6661
```

Two levels of nested functions:

Two levels of nested functions are created when nested functions are written inside nested functions. The following shows an example for the format of a user-defined function with four nested functions in two levels.

```
function y=A(a1,a2)                  (Primary function A.)
.......
   function z=B(b1,b2)               (B is nested function in A.)
   .......
         function w=C(c1,c2)         (C is nested function in B.)
         .......
         end
   end
   function u=D(d1,d2)               (D is nested function in A.)
   .......
         function h=E(e1,e2)         (E is nested function in D.)
         .......
         end
   end
.......
end
```

The following rules apply to nested functions:

- A nested function can be called from a level above it. (In the preceding example, function A can call B or D, but not C or E.)
- A nested function can be called from a nested function at the same level within the primary function. (In the preceding example, function B can call D, and D can call B.)
- A nested function can be called from a nested function at any lower level.
- A variable defined in the primary function is recognized and can be redefined by a function that is nested at any level within the primary function.
- A variable defined in a nested function is recognized and can be redefined by any of the functions that contain the nested function.

7.12 Examples of MATLAB Applications

Sample Problem 7-5: Exponential growth and decay

A model for exponential growth or decay of a quantity is given by

$$A(t) = A_0 e^{kt}$$

where $A(t)$ and A_0 are the quantity at time t and time 0, respectively, and k is a constant unique to the specific application.

Write a user-defined function that uses this model to predict the quantity $A(t)$ at time t from knowledge of A_0 and $A(t_1)$ at some other time t_1. For function name and arguments, use `At = expGD(A0,At1,t1,t)`, where the output argument `At` corresponds to $A(t)$, and for input arguments, use `A0,At1,t1,t,` corresponding to A_0, $A(t_1)$, t_1, and t, respectively.

Use the function file in the Command Window for the following two cases:

(*a*) The population of Mexico was 67 million in the year 1980 and 79 million in 1986. Estimate the population in 2000.

(*b*) The half-life of a radioactive material is 5.8 years. How much of a 7-gram sample will be left after 30 years?

Solution

To use the exponential growth model, the value of the constant k has to be determined first by solving for k in terms of A_0, $A(t_1)$, and t_1:

$$k = \frac{1}{t_1}\ln\frac{A(t_1)}{A_0}$$

Once k is known, the model can be used to estimate the population at any time.

The user-defined function that solves the problem is:

```
function At=expGD(A0,At1,t1,t)
% expGD calculates exponential growth and decay
% Input arguments are:
% A0: Quantity at time zero.
% At1: Quantity at time t1.
% t1: The time t1.
% t: time t.
% Output argument is:
% At: Quantity at time t.
k=log(At1/A0)/t1;
At=A0*exp(k*t);
```

Function definition line.

Determination of k.

Determination of $A(t)$.
(Assignment of value to output variable.)

Once the function is saved, it is used in the Command Window to solve the two cases. For case *a*) $A_0 = 67$, $A(t_1) = 79$, $t_1 = 6$, and $t = 20$:

```
>> expGD(67,79,6,20)
ans =
       116.03
```

Estimation of the population in the year 2000.

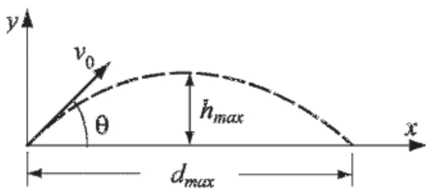
For case *b*) $A_0 = 7$, $A(t_1) = 3.5$ (since t_1 corresponds to the half-life, which is the time required for the material to decay to half of its initial quantity), $t_1 = 5.8$, and $t = 30$.

```
>> expGD(7,3.5,5.8,30)
ans =
         0.19
```

The amount of material after 30 years.

Sample Problem 7-6: Motion of a projectile

Create a function file that calculates the trajectory of a projectile. The inputs to the function are the initial velocity and the angle at which the projectile is fired. The outputs from the function are the maximum height and distance. In addition, the function generates a plot of the trajectory. Use the function to calculate the trajectory of a projectile that is fired at a velocity of 230 m/s at an angle of 39°.



Solution

The motion of a projectile can be analyzed by considering the horizontal and vertical components. The initial velocity v_0 can be resolved into horizontal and vertical components

$$v_{0x} = v_0\cos(\theta) \quad \text{and} \quad v_{0y} = v_0\sin(\theta)$$

In the vertical direction the velocity and position of the projectile are given by:

$$v_y = v_0 - gt \quad \text{and} \quad y = v_{0y}t - \frac{1}{2}gt$$

The time it takes the projectile to reach the highest point ($v_y = 0$) and the corresponding height are given by:

$$t_{h\max} = \frac{v_{0y}}{g} \quad \text{and} \quad h_{h\max} = \frac{v_{0y}^2}{2g}$$

The total flying time is twice the time it takes the projectile to reach the highest point, $t_{tot} = 2t_{hmax}$. In the horizontal direction the velocity is constant, and the position of the projectile is given by:

$$x = v_{0x}t$$

In MATLAB notation the function name and arguments are entered as `[hmax,dmax] = trajectory(v0,theta)`. The function file is:

```
function [hmax,dmax]=trajectory(v0,theta)
% trajectory calculates the max height and distance of a
projectile, and makes a plot of the trajectory.
% Input arguments are:
% v0: initial velocity in (m/s).
% theta: angle in degrees.
% Output arguments are:
% hmax: maximum height in (m).
% dmax: maximum distance in (m).
% The function creates also a plot of the trajectory.
g=9.81;
v0x=v0*cos(theta*pi/180);
v0y=v0*sin(theta*pi/180);
thmax=v0y/g;
hmax=v0y^2/(2*g);
ttot=2*thmax;
dmax=v0x*ttot;
% Creating a trajectory plot
tplot=linspace(0,ttot,200);
x=v0x*tplot;
y=v0y*tplot-0.5*g*tplot.^2;
plot(x,y)
xlabel('DISTANCE (m)')
ylabel('HEIGHT (m)')
title('PROJECTILE''S TRAJECTORY')
```

Function definition line.

Creating a time vector with 200 elements.

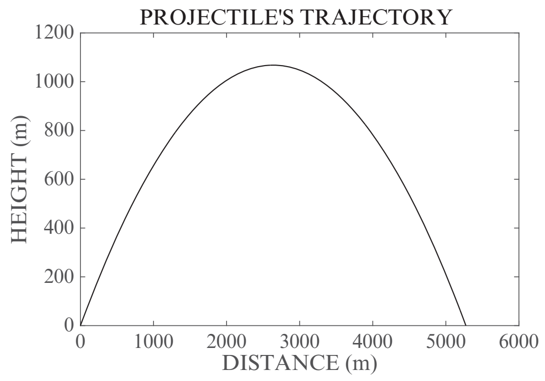
Calculating the x and y coordinates of the projectile at each time.

Note the element-by-element multiplication.

After the function is saved, it is used in the Command Window for a projectile that is fired at a velocity of 230 m/s and an angle of 39°.

```
>> [h d]=trajectory(230,39)
h =
  1.0678e+003
d =
  5.2746e+003
```

In addition, the following figure is created in the Figure Window:



7.13 *Problems*

1. Write a user-defined MATLAB function for the following math function:

$$y(x) = 0.6x^3 e^{-0.47x} + 1.5x^2 e^{-0.6x}$$

The input to the function is x and the output is y. Write the function such that x can be a vector (use element-by-element operations).
(*a*) Use the function to calculate $y(-2)$ and $y(4)$.
(*b*) Use the function to make a plot of the function $y(x)$ for $-4 \le x \le 8$.

2. Write a user-defined MATLAB function for the following math function:

$$r(\theta) = 3\sin(3\cos(0.5\theta))$$

The input to the function is θ (in radians) and the output is r. Write the function such that θ can be a vector.
(*a*) Use the function to calculate $r(\pi/6)$ and $r(5\pi/6)$.
(*b*) Use the function to plot (polar plot) $r(\theta)$ for $0 \le \theta \le 4\pi$.

3. In the U.S. fuel efficiency of cars is specified in miles per gallon (mpg). In Europe it is often expressed in liters per 100 km. Write a MATLAB user-defined function that converts fuel efficiency from mpg to liters per 100 km. For the function name and arguments, use `Lkm=mpgToLpkm(mpg)`. The input argument `mpg` is the efficiency in mi/gl, and the output argument `Lkm` is the efficiency in liters per 100 km (rounded to the nearest hundredth). Use the function in the Command Window to:
(*a*) Determine the fuel efficiency in liters per 100 km of a car whose fuel efficiency is 21 mi/gal.
(*b*) Determine the fuel efficiency in liters per 100 km of a car whose fuel efficiency is 36 mi/gal.

4. Pressure in U.S. customary units is measured in psi (pound per square inch). In SI metric units pressure is measured in Pa (N/m^2). Write a user-defined MATLAB function that converts pressure given in units of psi to pressure in units of Pa. For the function name and arguments, use `[Pa] = PsiToPa(psi)`. The input argument `psi` is the pressure in units of psi to be converted, and the output argument `Pa` is the converted pressure in units of Pa (rounded to the nearest integer). Use the function in the Command Window to:
   (*a*) Convert 120 psi to units of Pa.
   (*b*) Convert 3,000 psi to units of Pa.

5. Tables of material properties list density, in units of kg/m^3, when the international system of units (SI) is used, and list specific weight, in units of lb/in.^{3}, when the U.S. customary system of units is used. Write a user-defined MATLAB function that converts density to specific weight. For the function name and arguments, use `[sw] = DenToSw(den)`. The input argument `den` is the density of a material in kg/m^3, and the output argument `sw` is the specific weight in lb/in.^{3}. Use the function in the Command Window to:
   (*a*) Determine the specific weight of copper whose density is 8,960 kg/m^3.
   (*b*) Determine the specific weight of concrete whose density is 2,340 kg/m^3.

6. Write a user-defined MATLAB function that converts torque given in units of N-m to torque in units of lb-ft. For the function name and arguments, use `lbft = NmTOlbft(Nm)`. The input argument `Nm` is the torque in N-m, and the output argument `lbft` is the torque in lb-ft (rounded to the nearest integer). Use the function to convert 2,000 N-m to units of lb-ft.

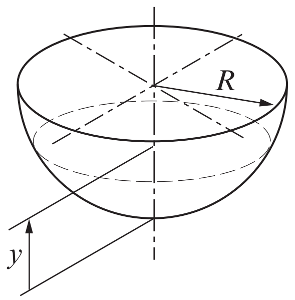
7. The body surface area (*BSA*) in m^2 of a person (used for determining dosage of medications) can be calculated by the formula (Mosteller formula):

$$BSA = \sqrt{H \times W / 3131}$$

in which *H* is the person's height in inches, and *W* is the persons weight in lb.
   Write a MATLAB user-defined function that calculates the body surface area. For the function name and arguments, use `BSA = BodySurA(w,h)`. The input arguments `w` and `h` are the weight and height, respectively. The output argument `BSA` is the *BSA* value. Use the function to calculate the body surface area of:
   (*a*) A 170-lb, 5-ft 10-in. tall person.
   (*b*) A 220-lb, 6-ft 5-in. tall person.

8. The fuel tank shown in the figure in shaped as a half a sphere with $R = 24$ in.

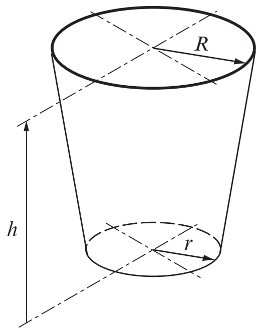
   Write a user-defined function that calculates the volume of fuel in the tank (in gallons) as a function of the height y (measured from the bottom). For the function name and arguments, use `V = Volfuel(y)`. Use the function to make a plot of the volume as a function of y for $0 \le y \le 24$ in.



9. A paper cup is designed to have a geometry of a frustum of a cone. Write a user-defined function that determines the volume and the surface area (side plus bottom) of the cup for given values of r, R, and h. For the function name and arguments, use `[V, S] = VolSArea(r,R,h)`. The input arguments `r`, `R`, and `h` are the radius of the base, the radius of the top and the height, respectively (all in units of inches). The output arguments `V` and `S` are the volume (in units of U.S. fluid ounce) and the surface area (in units of in.^{2}), respectively. Use the function to determine the volume and the surface area of cups with the following dimensions:

   (*a*) $r = 2$ in., $R = 3.5$ in., $h = 4.25$ in.

   (*b*) $r = 2.5$in., $R = 3.5$ in., $h = 4.5$ in.



10. The relative humidity, RH, at sea level can be calculated from measured values of the dry-bulb temperature, T_{db}, and the wet-bulb temperature T_{wb} by (temperatures in degrees Celsius):

$$RH = \frac{VP}{SVP}100$$

where VP is the vapor pressure given by:

$$VP = e^{\frac{16.78T_{wb}-116.9}{T_{wb}+237.3}} - 0.066858(1+0.00115T_{wb})(T_{db}-T_{wb})$$

and SVP is the saturated vapor pressure given by:

$$SVP = e^{\frac{16.78T_{db}-116.9}{T_{db}+237.3}}$$

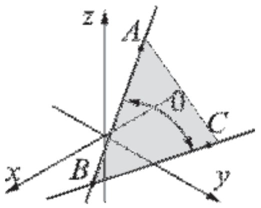
Write a user-defined function for calculating RH for given T_{db} and T_{wb}. For the function name and arguments, use `RH = RelHum(Tdb,Twb)`. The input arguments are `Tdb` and `Twb` are the dry-bulb and wet-buld temperatures, respectively in °F. The output argument `RH` is the relative humidity in percent (rounded to the nearest integer). Inside the user-defined function use a subfunction, or an anonymous function to convert the unit of the temperature from Cesius to Fahrenheit. Use the function to determine the relative humidity for the following conditions:

(*a*) $T_{db} = 75°$F, $T_{wb} = 69°$F. (*b*) $T_{db} = 93°$F, $T_{wb} = 90°$ F.

11. Write a user-defined function that calculates grade point average (GPA) on a scale of 0 to 5, where $A = 5$, $B = 4$, $C = 3$, $D = 2$, and $F = 0$. For the function name and arguments, use `GPA = GradePtAve(G,C)`. The input argument `G` is a vector whose elements are letter grades *A*, *B*, *C*, *D*, or *F* entered as a string (e.g., `['ABACFB']`). The input argument `C` is a vector with the corresponding credit hours. The output argument `GPA` is the calculated GPA rounded to the nearest tenth (i.e., 3.75 is rounded to 3.8, and 3.749 is rounded to 3.7). Use the function to calculate the GPA for a student with the following record:

Grade	*A*	*B*	*F*	*C*	*B*	*A*	*D*	*A*
Credit Hours	4	3	3	2	3	4	3	3

12. Write a user-defined MATLAB function that determines the angle that forms by the intersection of two lines. For the function name and arguments, use `th=anglines(A,B,C)`. The input arguments to the function are vectors with the coordinates of the points *A*, *B*, and *C*, as shown in the figure, which can be two- or three-dimensional. The output `th` is the angle in degrees. Use the function `anglines` for determining the angle for the following cases:



(*a*) *A*(–5, –1, 6), *B*(2.5, 1.5, –3.5), *C*(–2.3, 8, 1)
(*b*) *A*(–5.5, 0), *B*(3.5, –6.5), *C*(0, 7)

13 Write a user-defined MATLAB function that determines the time elapsed between two events during a day. For the function name and arguments, use `dt = timediff(TA,ap1,TB,ap2)`. The input arguments to the function are:

`TA` is a two-element vector with the time of the first event. The first element is the hour and the second element is the minute.

`ap1` is a string `'AM'` or `'PM'` which corresponds to the time of the first event.

`TB` is a two-element vector with the time of the second event. The first element is the hour and the second element is the minute.

`ap2` is a string `'AM'` or `'PM'` which corresponds to the time of the second event.

The output argument `dt` is a two-element vector with the time elapsed between two events. The first element is the number of hours and the second element is number of minutes.

The function displays an error message if the time entered for event *B* is before the time entered for event *A*.

Use the function to determine the time elapsed between the following events:

(*a*) Event A: 5:37 AM; Event B: 2:51 PM.
(*b*) Event A: 12:53 PM; Event B: 6:12 PM.
(*c*) Event A: 11:32 PM; Event B: 3:18 PM. (Error situation.)

14. Write a user-defined MATLAB function that determines the unit vector in the direction of the line that connects two points (*A* and *B*) in space. For the function name and arguments, use `n = unitvec(A,B)`. The input to the function are two vectors `A` and `B`, each with the Cartesian coordinates of the corresponding point. The output `n` is a vector with the components of the unit vector in the direction from *A* to *B*. If points *A* and *B* have two coordinates each (they are in the *x y* plane), then `n` is a two-element vector. If points *A* and *B* have three coordinate each (general points in space), then `n` is a three-element vector. Use the function to determine the following unit vectors:
(*a*) In the direction from point (–0.7, 2.1) to point (9, 18).
(*b*) In the direction from point (10, –3.5, –2.5) to point (–11, 6.5, 5.9).

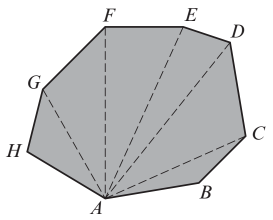
15. Write a user-defined MATLAB function that determines the cross product of two vectors. For the function name and arguments, use `w=crosspro(u,v)`. The input arguments to the function are the two vectors, which can be two- or three-dimensional. The output `w` is the result (a vector). Use the function `crosspro` for determining the cross product of:
(*a*) Vectors $a = 3i + 11j$ and $b = 14i - 7.3j$.
(*b*) Vectors $c = -6i + 14.2j + 3k$ and $d = 6.3i - 8j - 5.6k$.

16. The area of a triangle *ABC* can be calculated by:

$$A = \frac{1}{2}|\boldsymbol{AB} \times \boldsymbol{AC}|$$

where ***AB*** is the vector from vertex *A* to vertex *B* and ***AC*** is the vector from vertex *A* to vertex *C*. Write a user-defined MATLAB function that determines the area of a triangle given its vertices' coordinates. For the function name and arguments, use `[Area] = TriArea(A,B,C)`. The input arguments `A`, `B`, and `C` are vectors, each with the coordinates of the corresponding vertex. Write the code of `TriArea` such that it has two subfunctions—one that determines the vectors ***AB*** and ***AC*** and another that executes the cross product. (If available, use the user-defined functions from Problem 15). The function should work for a triangle in the *x*-*y* plane (each vertex is defined by two coordinates) or for a triangle in space (each vertex is defined by three coordinates). Use the function to determine the areas of triangles with the following vertices:
(*a*) $A = (1, 2)$, $B = (10, 3)$, $C = (6, 11)$
(*b*) $A = (-1.5, -4.2, -3)$, $B = (-5.1, 6.3, 2)$, $C = (12.1, 0, -1.5)$

17. As shown in the figure, the area of a convex polygon can be calculated by adding the area of the triangles that the polygon can be divided into. Write a user-defined MATLAB function that calculates the area of a convex *n*-sided polygon. For the function name and arguments, use `A = APolygon(Crd)`. The input argument `Crd` is a two-column matrix where each row contains the coordinates of a vertex (first column is the *x* coordinate and the second column is the *y* coordinate). The vertices are listed in the order that they are connected to form the polygon (i.e., coordinates of point *A* in the first row, point *B* in the second, and so on). The output argument `A` is the area of the polygon. Write the code of `APolygon` such that it has a subfunction that calculates the area of a triangle for given vertices' coordinates. Use `APolygon` to calculate the area of the polygon shown in the figure. The coordinates of the vertices are: *A*(1, 1), *B*(7, 2), *C*(10, 5), *D*(9, 11), *E*(6, 12), *F*(1, 12), *G*(–3, 8), *H*(–4, 4).

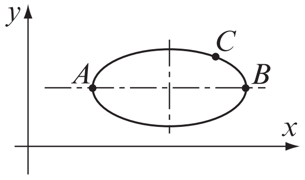


18. Write a user-defined function that determines the location of the center and the radius of a circle that passes through three given points in a plane. The function also creates a plot that shows the circle and the points. For the function name and arguments, use `[C R]=Circle3Pts(A,B,C)`. The input arguments `A`, `B`, and `C` are each a two-element vector with the *x* and *y* coordinates of the corresponding point. The output argument `C`, is a vector with the coordinates of the center the output argument `R`, is the radius (both rounded to the nearest hundredth). Use the function with the following three points: *A*(7, 1.2), *B*(0.5, 2.6), and *C*(–2.4, –1.4).

19. Write a user-defined MATLAB function that converts integers written in decimal form to binary form. Name the function `b=Bina(d)`, where the input argument `d` is the integer to be converted and the output argument `b` is a vector with 1s and 0s that represents the number in binary form. The largest number that could be converted with the function should be a binary number with 16 1s. If a larger number is entered as `d`, the function should display an error message. Use the function to convert the following numbers:
(*a*) 100 (*b*) 1,002 (*c*) 52,601 (*d*) 200,090

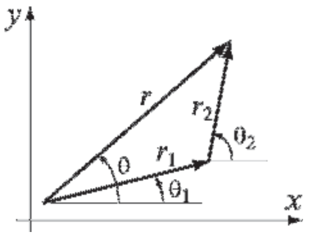
20. Write a user-defined function that plots a triangle and the circle that is inscribed inside, given the coordinates of its vertices. For the function name and arguments, use `TriCirc(A,B,C)`. The input arguments are vectors with the *x* and *y* coordinates of the vertices, respectively. This function has no output arguments. Use the function with the points (2.6, 3.2), (11, 14.5), and (–2, 2.8).

21. Write a user-defined function that plots an ellipse with axes that are parallel to the x and y axes, given the coordinates of its vertices and the coordinates of another point that the ellipse passes through. For the function name and arguments, use `ellipseplot(A,B,C)`. The input arguments `A` and `B` are each a two-element vector with the coordinates of the vertices, and `C` is a two-element vector with the coordinates of another point on the ellipse (see figure), respectively. This function has no output arguments. Use the function to plot the following ellipses:



(*a*) $A(2,3)$, $B(11,3)$, $C(10,4)$ (*b*) $A(2,11)$, $B(2,-4)$, $C(4,8)$

22. In polar coordinates a two-dimensional vector is given by its radius and angle (r, θ). Write a user-defined MATLAB function that adds two vectors that are given in polar coordinates. For the function name and arguments, use
`[r th]=AddVecPol(r1,th1,r2,th2)`,
where the input arguments are (r_1, θ_1) and (r_2, θ_2), and the output arguments are the radius and angle of the result. Use the function to carry out the following additions:

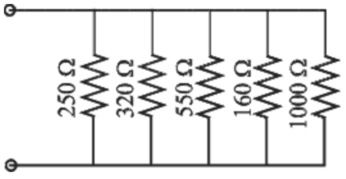


(*a*) $r_1 = (5, 23°)$, $r_2 = (12, 40°)$ (*b*) $r_1 = (6, 80°)$, $r_2 = (15, 125°)$

23. Write a user-defined function that determines if a number is a prime number. Name the function `pr=Trueprime(m)`, where the input arguments `m` is a positive integer and the output argument `pr` is 1 if `m` is a prime number and 0 if `m` is not a prime number. Do not use MATLAB's built-in functions `primes` and `isprime`. If a negative number or a number that is not an integer is entered when the function is called, the error message "The input argument must be a positive integer." is displayed.
(*a*) Use the function with 733, 2001, and 107.5.
(*b*) Write a MATLAB program in a script file that makes use of `Trueprime` and finds the smallest prime number that remains a prime number when added to its reverse (37 is the reverse of 73).

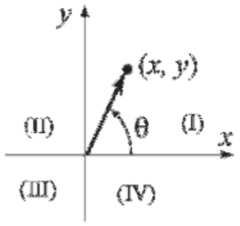
24. The harmonic mean H of a set of n positive numbers $x_1, x_2, \ldots, x_n$ is defined by:

$$H = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \ldots + \frac{1}{x_n}}$$



Write a user-defined function that calculates the harmonic mean of a set of numbers. For function name and arguments use `G=Harmean(x)`, where the input argument `x` is a vector of numbers (any length) and the output argument `H` is their harmonic mean. In electrical engineering the equivalent resistance of resistors connected in parallel is equal to the harmonic mean of the values of the resistors divided by the number of the resistors. Use the user-defined function `Harmean` to calculate the equivalent resistance of the resistors shown in the figure.

25. Write a user-defined function that determines the polar coordinates of a point from the Cartesian coordinates in a two-dimensional plane. For the function name and arguments, use `[th rad]=CartToPolar(x,y)`. The input arguments are the x and y coordinates of the point, and the output arguments are the angle θ and the radial distance to the point. The angle θ is in degrees and is measured relative to the positive x axis, such that it is a positive number in quadrants I and II, and a negative number in quadrant III and IV. Use the function to determine the polar coordinates of points (14, 9), (–11, –20), (–15, 4), and (13.5, –23.5).



26. Write a user-defined function that determines the value that occurs most often in a set of data that is given in a two-dimensional matrix. For the function name and arguments, use `[v, q] =matrixmode(x)`. The input argument `x` is a $m \times n$ matrix of any size with numerical values, and the output arguments `v` and `q` are the values that occur most often and the number of times they occur. If there are two, or more, values that occur most often than `v` is a vector with these values. Do not use the MATLAB built-in function `mode`. Test the function three times. For input create a 5×6 matrix using the following command: `x=randi(10,5,6)`.

27. Write a user-defined function that sorts the elements of a vector from the largest to the smallest. For the function name and arguments, use `y=downsort(x)`. The input to the function is a vector `x` of any length, and the output `y` is a vector in which the elements of `x` are arranged in a descending order. Do not use the MATLAB built-in functions `sort`, `max`, or `min`. Test your function on a vector with 14 numbers (integers) randomly distributed between –30 and 30. Use the MATLAB `randi` function to generate the initial vector.

28. Write a user-defined function that sorts the elements of a matrix. For the function name and arguments, use `B = matrixsort(A)`, where `A` is any size $(m \times n)$ matrix and `B` is a matrix of the same size with the elements of `A` rearranged in descending order column after column with the (1,1) element the largest and the (m,n) element the smallest. If available, use the user-defined function `downsort` from the previous problem as a subfunction

within `matrixsort`.

Test your function on a 4×7 matrix with elements (integers) randomly distributed between –30 and 30. Use MATLAB's `randi` function to generate the initial matrix.

29. Write a user-defined MATLAB function that finds the largest element of a matrix. For the function name and arguments, use `[Em,rc] = matrixmax(A)`, where `A` is any size matrix. The output argument `Em` is the value of the largest element, and `rc` is a two-element vector with the address of the largest element (row and column numbers). If there are two, or more, elements that have the maximum value, the output argument `rc` is a two-column matrix where the rows list the addresses of the elements. Test the function three times. For input create a 4×6 matrix using the following command: `x=randi([-20 100],4,6)`.

30. Write a user-defined MATLAB function that calculates the determinant of a 3×3 matrix by using the formula:

$$det = A_{11}\begin{vmatrix} A_{22} & A_{23} \\ A_{32} & A_{33} \end{vmatrix} - A_{12}\begin{vmatrix} A_{21} & A_{23} \\ A_{31} & A_{33} \end{vmatrix} + A_{13}\begin{vmatrix} A_{21} & A_{22} \\ A_{31} & A_{32} \end{vmatrix}$$

For the function name and arguments, use `d3 = det3by3(A)`, where the input argument `A` is the matrix and the output argument `d3` is the value of the determinant. Write the code of `det3by3` such that it has a subfunction that calculates the 2×2 determinant. Use `det3by3` for calculating the determinants of:

(*a*) $\begin{bmatrix} 1 & 3 & 2 \\ 6 & 5 & 4 \\ 7 & 8 & 9 \end{bmatrix}$ (*b*) $\begin{bmatrix} -2.5 & 7 & 1 \\ 5 & -3 & -2.6 \\ 4 & 2 & -1 \end{bmatrix}$

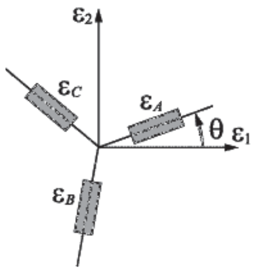
31. The shortest distance between two points on the surface of the globe (great-circle distance) can be calculated by using the haversine formula. If Φ_1 and λ_1 are the latitude and longitude of point 1 and Φ_2 and λ_2 are the latitude and longitude of point 2, the great circle distance between the points is given by:

$$d = 2R\sin^{-1}\left(\sqrt{a}\right)$$

where $a = \sin^2\left(\frac{\Phi_2 - \Phi_1}{2}\right) + \cos\Phi_1\cos\Phi_2\cos\Phi_1\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)$, and $R = 3,959$ mi is the Earth radius. Write a user-defined function that determines the distance between two points on the Earth. For the function name and arguments, use `dis = GreatCirDis(Lat1,Lng1,Lat2,Lng2)`, where the input arguments are the latitude and longitude of the two points (degrees in decimal format), and `dis` is the great-circle distance in miles. Use the function to calculate the distance between London (51.50853°, –0.12574°) and New

York City (40.71427°, –74.00597°).

32. Delta rosette is a set of three strain gages oriented at 120° relative to each other. The strain measured with each of the strain gages is ε_A, ε_B, and ε_C. The principal strains ε_1 and ε_2 can be calculated from the strains measured with the rosette by:



$$\epsilon_{1,2} = \frac{\epsilon_A+\epsilon_B+\epsilon_C}{3} \pm \frac{\sqrt{2}}{3}\sqrt{(\epsilon_A-\epsilon_B)^2+(\epsilon_B-\epsilon_C)^2+(\epsilon_C-\epsilon_A)^2}$$

Write a user-defined MATLAB function that determines the principal strains given the strains ε_A, ε_B, and ε_C. For the function name and arguments, use `[P1, P2]=DeltaRos(A,B,C)`. The input arguments `A`, `B`, and `C` are the values of the three strains measured by the rosette. The output arguments `P1` and `P2` are the values of the principal strains.

Use the function to determine the principal strains for the following cases:

(*a*) $\epsilon_A = 42\ \mu$, $\epsilon_B = 970\ \mu$, $\epsilon_C = 340\ \mu$.

(*b*) $\epsilon_A = 110\ \mu$, $\epsilon_B = 80\ \mu$, $\epsilon_C = -60\ \mu$.

33. In a lottery the player has to select several numbers out of a list. Write a user-defined function that generates a list of *n* integers that are uniformly distributed between the numbers *a* and *b*. All the selected numbers on the list must be different. For function name and arguments, use `x=lotto(a,b,n)` where the input arguments are the numbers *a* and *b*, and *n*, respectively. The output argument `x` is a vector with the selected numbers.

(*a*) Use the function to generate a list of seven numbers from the numbers 1 through 59.

(*b*) Use the function to generate a list of eight numbers from the numbers 50 through 65.

(*c*) Use the function to generate a list of nine numbers from the numbers –25 through –2.

34. The Taylor series expansion for $\sin x$ about $x = 0$ is given by:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^{57}}{7!} + \ldots = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

where x is in radians. Write a user-defined function that determines $\sin x$ using Taylor's series expansion. For function name and arguments, use `y=sinTay(x)`, where the input argument `x` is the angle in degrees and the output argument `y` is the value of $\sin x$. Inside the user-defined function, use a loop for adding the terms of the Taylor series. If a_n is the *n*th term in the series, then the sum S_n of the *n* terms is $S_n = S_{n-1} + a_n$. In each pass, calculate the estimated error E given by $E = \left|\frac{S_n - S_{n-1}}{S_{n-1}}\right|$. Stop adding terms when

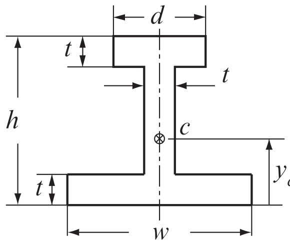
$E \le 0.000001$. Since $\sin\theta = \sin(\theta \pm 360n)$ (n is an integer) write the user-defined function such that if the angle is larger than 360°, or smaller than –360°, then the Taylor series will be calculated using the smallest number of terms (using a value for x that is closest to 0).

Use `sinTay` for calculating:

(*a*) $\sin 39°$ (*b*) $\sin 205°$ (*c*) $\sin(-70°)$.
(*d*) $\sin 754°$ (*e*) $\sin 19{,}000°$ (*f*) $\sin(-748°)$

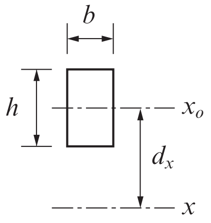
Compare the values calculated using `sinTay` with the values obtained by using MATLAB's built-in `sind` function.

35. Write a user-defined function that determines the coordinate y_c of the centroid of the I-shaped cross-sectional area shown in the figure. For the function name and arguments, use `yc = centroidI(w,h,d,t)`, where the input arguments `w`, `h`, `d`, and `t` are the dimensions shown in the figure and the output argument `yc` is the coordinate y_c.

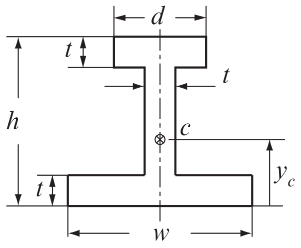


Use the function to determine y_c for a beam with $w = 10$ in., $h = 8$ in., $d = 6$ in., and $t = 0.5$ in.

36. The area moment of inertia I_{x_o} of a rectangle about the axis x_o passing through its centroid is $I_{x_o} = \frac{1}{12}bh^3$. The moment of inertia about an axis x that is parallel to x_o is given by $I_x = I_{x_o} + Ad_x^2$, where A is the area of the rectangle, and d_x is the distance between the two axes.



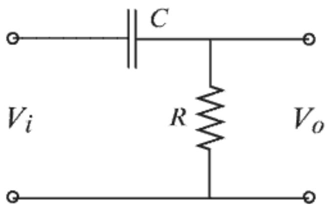
Write a MATLAB user-defined function that determines the area moment of inertia I_{x_c} of a I-beam about the axis that passes through its centroid (see drawing). For the function name and arguments use `Ixc=IxcBeam(w,h,d,t)`, where the input arguments `w`, `h`, `d`, and `t` are the dimensions shown in the figure and the output argument `Ixc` is I_{x_c}. For finding the coordinate y_c of the centroid, use the user-defined function `centroidI` from the previous problem as a subfunction inside `IxcBeam`.



(The moment of inertia of a composite area is obtained by dividing the area into parts and adding the moments of inertia of the parts.)

Use the function to determine the moment of inertia for a beam with $w = 10$ in., $h = 8$ in., $d = 6$ in., and $t = 0.5$ in.

37. The simple *RC* high-pass filter shown in the figure passes signals with frequencies higher than a certain cutoff frequency. The ratio of the magnitudes of the voltages is given by:



$$RV = \left|\frac{V_o}{V_i}\right| = \frac{\omega RC}{\sqrt{1+\omega^2R^2C^2}}$$

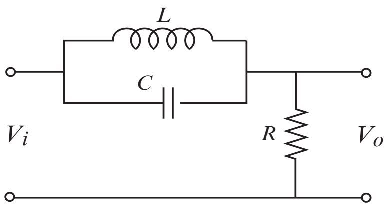
where $\omega = 2\pi f$, and f is the frequency of the input signal.

Write a user-defined MATLAB function that calculates the ratio of magnitudes for given values of *R*, *C*, and *f*. For the function name and arguments, use `RV = RCFilt(R,C,f)`. The input arguments are `R`, the size of the resistor in Ω (ohms); `C`, the size of the capacitor in F (farad); and `f`, the frequency of the input signal in Hz (hertz). Write the function such that `f` can be a vector.

Write a program in a script file that uses the `RCFilt` function to generate a plot of *RV* as a function of *f* for $10 \le f \le 100,000$ Hz. The plot has a logarithmic scale on the horizontal axis. When executed, the script file asks the user to enter the values of *R* and *C*. Label the axes of the plot.

Run the script file with $R = 80$ Ω, and $C = 5$ μF.

38. A circuit that filters out a certain frequency is shown in the figure. In this filter, the ratio of the magnitudes of the voltages is given by:



$$RV = \left|\frac{V_o}{V_i}\right| = \frac{|R(1-\omega^2LC)|}{\sqrt{(\omega L)^2+(R-R\omega^2LC)^2}}$$

where $\omega = 2\pi f$, and f is the frequency of the input signal.

Write a user-defined MATLAB function that calculates the ratio of magnitudes. For the function name and arguments, use `RV=filtafreq(R,C,L,f)`. The input arguments are `R` the size of the resistor in Ω (ohms); `C`, the size of the capacitor in F (farad); `L`, the inductance of the coil in H (henrys); and `f`, the frequency of the input signal in Hz (hertz). Write the function such that `f` can be a vector.

Write a program in a script file that uses the `filtafreq` function to generate a figure with two plots of *RV* as a function of *f* for $10 \le f \le 100,000$ Hz. In one plot $C = 160$ μF, $L = 45$ mH, and $R = 200$ Ω, and in the second plot *C* and *L* are unchanged but $R = 50$ Ω. The plot has a logarithmic scale on the horizontal axis. Label the axes and display a legend.

39. The first derivative $\frac{df(x)}{dx}$ of a function $f(x)$ at a point $x = x_0$ can be approximated with the four-point central difference formula:

$$\frac{df}{dx} = \frac{f(x_0-2h)-8f(x_0-h)+8f(x_0+h)-f(x_0+2h)}{12h}$$

where h is a small number relative to x_0. Write a user-defined function function (see Section 7.9) that calculates the derivative of a math function $f(x)$ by using the four-point central difference formula. For the user-defined function name, use `dfdx=FoPtder(Fun,x0)`, where `Fun` is a name for the function that is passed into `FoPtder`, and `x0` is the point where the derivative is calculated. Use $h = x_0 / 100$ in the four-point central difference formula. Use the user-defined function `FoPtder` to calculate the following:

(*a*) The derivative of $f(x) = x^3e^{2x}$ at $x_0 = 0.6$.

(*b*) The derivative of $f(x) = \frac{3^x}{x^2}$ at $x_0 = 2.5$.

In both cases compare the answer obtained from `FoPtder` with the analytical solution (use `format long`).

40. In lottery the player has to guess correctly *r* numbers that are drawn out of *n* numbers. The probability, *P*, of guessing *m* numbers out of the *r* numbers can be calculated by the expression:

$$P = \frac{C_{r,m}C_{(n-r),(r-m)}}{C_{n,r}}$$

where $C_{x,y} = \frac{x!}{y!(x-y)!}$. Write a user-defined MATLAB function that calculates *P*. For the function name and arguments, use `P = ProbLottery(m,r,n)`. The input arguments are `m`, the number of correct guesses; `r`, the number of numbers that need to be guessed; and `n`, the number of numbers available. Use a subfunction inside `ProbLottery` for calculating $C_{x,y}$.

(*a*) Use `ProbLottery` for calculating the probability of correctly selecting 3 of 6 numbers that are drawn out of 49 numbers in a lottery game.

(*b*) Consider a lottery game in which 6 numbers are drawn out of 49 numbers. Write a program in a script file that displays a table with seven rows and two columns. The first column has the numbers 0, 1, 2, 3, 4, 5, and 6, which are the number of numbers guessed correctly. The second column show the corresponding probability of making the guess.