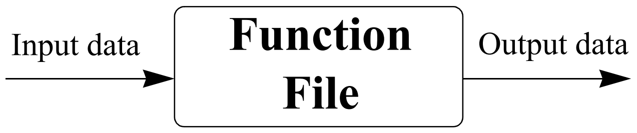
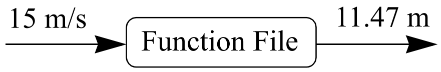
Chapter 7
User-Defined Functions and Function Files

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 can be 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, vector, or an array of any size. Schematically, a function file can be illustrated by:



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.



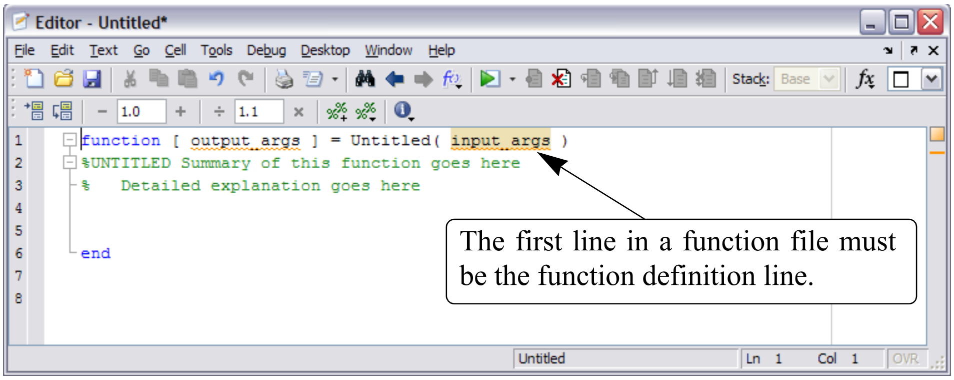
In addition to being used as math functions, user-defined functions can be used as subprograms in large programs. In this way large computer programs can be made up of smaller "building blocks" that can be tested independently. Function files are similar to subroutines in Basic and Fortran, procedures 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 and/or inline functions, which are 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 **File** menu, select **New**, and 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 selecting **Script** after **New**. The window that opens is empty, without any pre-typed lines. The window can be used for writing a script file or a



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


end
```

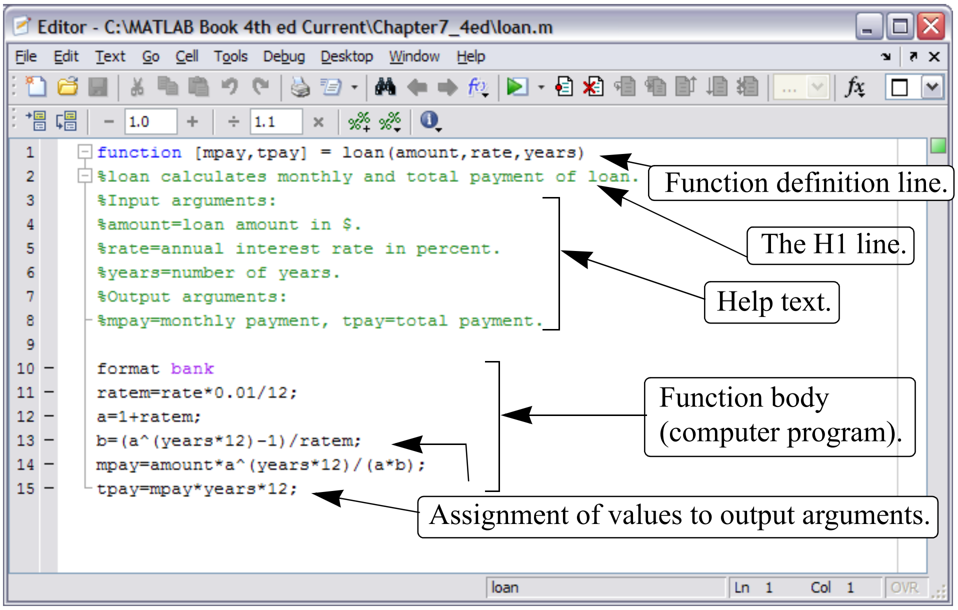
Editor - Untitled*
File Edit Text Go Cell Tools Debug Desktop Window Help
The first line in a function file must be the function definition line.
Untitled Ln 1 Col 1 OVR

Figure 7-1: The Editor/Debugger Window.

function file. If the Editor/Debugger Window is opened by selecting **Function** after **New**, it can also 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 4th ed Current\Chapter7_4ed\loan.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Function definition line.
The H1 line.
Help text.
Function body (computer program).
Assignment of values to output arguments.
loan Ln 1 Col 1 OVR

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

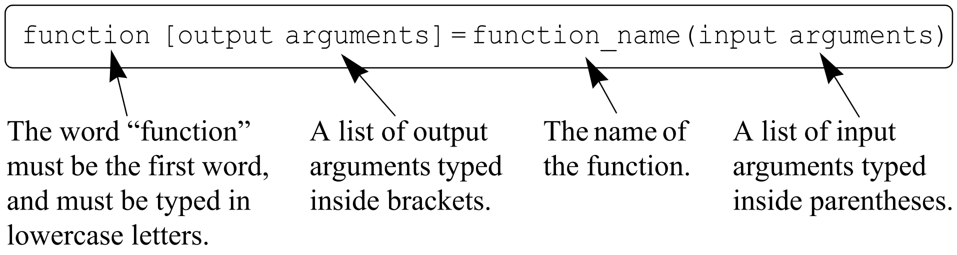
The various parts of the function file are described in detail in the following sections.

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. *In order 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 of 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)
```

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

```
month =
        600.72
total =
      28834.47
```

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;
```

Define variables `a` and `b`.

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

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 $f(x) = \frac{x^4\sqrt{3x+5}}{(x^2+1)^2}$. 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 ($\alpha = 23 \cdot 10^{-6}$ $1/°\mathrm{C}$) 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.

7.8 Anonymous and Inline Functions

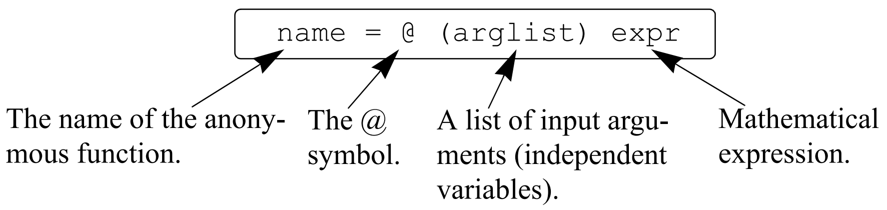
User-defined functions written in function files can be used for simple mathematical functions, for large and complicated math functions that require extensive programming, and as subprograms in large computer programs. In cases when the value of a relatively simple mathematical expression has to be determined 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).

Anonymous functions were introduced in MATLAB 7. They replace inline functions that were used for the same purpose in previous versions of MATLAB. Both anonymous and inline functions can be used in MATLAB R2010b). Anonymous functions, however, have several advantages over inline functions, and it is expected that inline functions will gradually be phased out. Anonymous functions are covered in detail in Section 7.8.1, and inline functions are described in the section that follows.

7.8.1 Anonymous 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 $f(x, y) = 2x^2 - 4xy + y^2$ 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
```

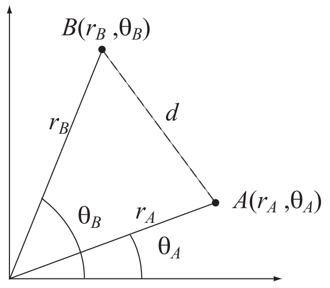
Then the anonymous function can be used for different values of *x* and *y*. For example, typing `HA(2,3)` gives:

```
>> 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:

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

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.
```

7.8.2 Inline Functions

Similar to an anonymous function, an inline function is a simple user-defined function that is defined without creating a separate function file (M-file). As already mentioned, anonymous functions replace the inline functions used in earlier versions of MATLAB. Inline functions are created with the `inline` command according to the following format:

```
name = inline('math expression typed as a string')
```

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

- The mathematical expression can have one or several independent variables.
- Any letter except i and j can be used for the independent variables in the expression.
- The mathematical expression can include any built-in or user-defined functions.
- The expression must be written according to the dimension of the argument (element-by-element or linear algebra calculations).
- The expression *cannot* include pre assigned variables.
- Once the function is defined it can be used by typing its name and a value for the argument (or arguments) in parentheses (see example below).
- The `inline` function can be used as an argument in other functions.

For example, the function: $f(x) = \dfrac{e^{x^2}}{\sqrt{x^2+5}}$ can be defined as an inline function for x by:

```
>> FA=inline('exp(x.^2)./sqrt(x.^2+5)')
FA =
     Inline function:
     FA(x) = exp(x.^2)./sqrt(x.^2+5)
>> FA(2)
ans =
   18.1994
>> FA([1 0.5 2])
ans =
    1.1097    0.5604   18.1994
```

Expression written with element-by-element operations.

Using a scalar as the argument.

Using a vector as the argument.

An inline function that has two or more independent variables can be written by using the following format:

```
name = inline('mathematical expression','arg1',
                                       'arg2','arg3')
```

In the format shown here the order of the arguments to be used when calling the function is defined. If the independent variables are not listed in the command, MATLAB arranges the arguments in alphabetical order. For example, the function $f(x, y) = 2x^2 - 4xy + y^2$ can be defined as an inline function by:

```
>> HA=inline('2*x^2-4*x*y+y^2')
HA =
     Inline function:
     HA(x,y) = 2*x^2-4*x*y+y^2
```

Once defined, the function can be used with any values of x and y. For example, HA(2,3) gives:

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

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 3×2 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);              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)=Fun((a+b)/2);          Using the imported function to
xyout(3,2)=y(100);                calculate f(x) at the midpoint.
plot(x,y)
xlabel('x'), ylabel('y')
```

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.

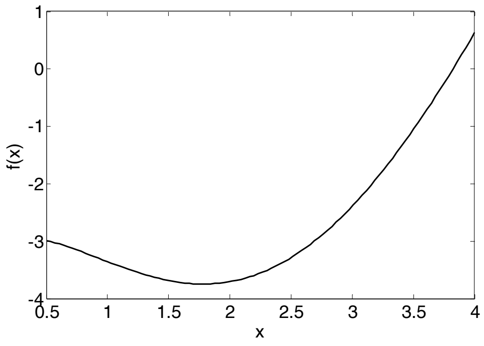


Figure 7-3: A plot of the function $f(x) = e^{-0.17x}x^3 - 2x^2 + 0.8x - 3$**.**

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
```

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

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

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 which 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` which was created earlier in the chapter (Figure 7-2). This function has three input arguments and two output arguments.

A $50,000 loan, 3.9% interest, 10 years.

```
>> [M,T]=feval('loan',50000,3.9,10)
M =
        502.22
T =
      60266.47
```

Monthly payment.

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);
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);
xyout(3,2)=y(100);
```

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

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

```
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.

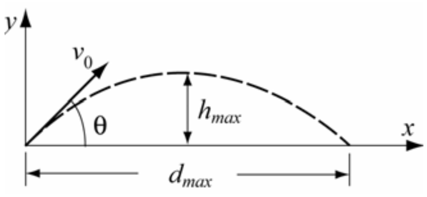
For case *b*) $A_0 = 7$, $A(t_1) = 3.5$ (since t_1 corresponds to the half-life, which is the time required for the material to decay to half of its initial quantity), $t_1 = 5.8$, and $t = 30$.

```
>> 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:

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

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

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

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)
```

Function definition line.

```
% 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);
```

Creating a time vector with 200 elements.

```
x=v0x*tplot;
y=v0y*tplot-0.5*g*tplot.^2;
```

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

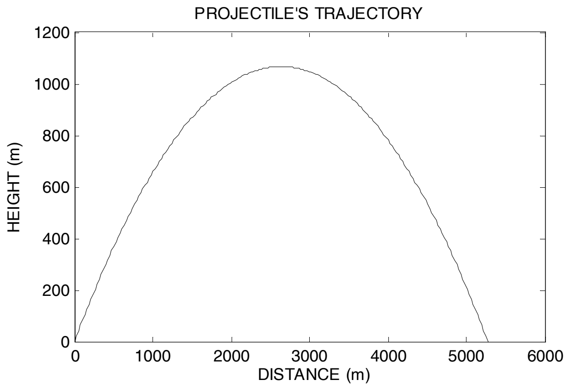
Note the element-by-element multiplication.

```
plot(x,y)
xlabel('DISTANCE (m)')
ylabel('HEIGHT (m)')
title('PROJECTILE''S TRAJECTORY')
```

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^o.

```
>> [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. The fuel efficiency of an automobile is measured in mi/gal (miles per U.S. gallon) or in km/L (kilometers per liter). Write a MATLAB user-defined function that converts fuel efficiency values from km/L to mi/gal. For the function name and arguments use `mpg=kmlTOmpg(kml)`. The input argument `kml` is the efficiency in km/L, and the output argument `mpg` is the efficiency in mi/gal. Use the function in the Command Window to:
   (*a*) Determine the fuel efficiency in mi/gal of a car that consumes 9 km/L.
   (b) Determine the fuel efficiency in mi/gal of a car that consumes 14 km/L.

2. Write a user-defined MATLAB function for the following math function:
$$y(x) = -0.2x^4 + e^{-0.5x}x^3 + 7x^2$$
   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 $y(-2.5)$, and $y(3)$.
   (*b*) Use the function to make a plot of the function $y(x)$ for $-3 \le x \le 4$.

3. Write a user-defined MATLAB function, with two input and two output arguments, that determines the height in centimeters and mass in kilograms of a person from his height in inches and weight in pounds. For the function name and arguments use `[cm, kg] = STtoSI(in,lb)`. The input arguments are the height in inches and weight in pounds, and the output arguments are the height in centimeters and mass in kilograms. Use the function in the Command Window to:
   (*a*) Determine in SI units the height and mass of a 5 ft 8 in. person who weighs 175 lb.
   (b) Determine your own height and weight in SI units.

4. Write a user-defined MATLAB function that converts speed given in units of miles per hour to speed in units of meters per second. For the function name and arguments use `mps = mphTOmets(mph)`. The input argument is the speed in mi/h, and the output argument is the speed in m/s. Use the function to convert 55 mi/h to units of m/s.

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

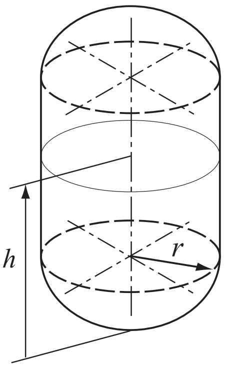
$$r(\theta) = 2\cos\theta\sin\theta\sin(\theta/4)$$

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 $r(3\pi/4)$ and $r(7\pi/4)$.
(*b*) Use the function to plot (polar plot) $r(\theta)$ for $0 \le \theta \le 2\pi$.

6. Write a user-defined MATLAB function that determines the area of a triangle when the lengths of the sides are given. For the function name and arguments use `[Area] = triangle(a,b,c)`. Use the function to determine the areas of triangles with the following sides:
(*a*) $a = 3$, $b = 8$, $c = 10$. (*b*) $a = 7$, $b = 7$, $c = 5$.

7. A cylindrical vertical fuel tank has hemispheric end caps as shown. The radius of the cylinder and the caps is $r = 15$ in., and the height of the cylindrical middle section is 40 in.

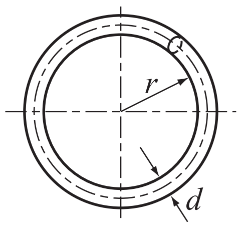
Write a user-defined function (for the function name and arguments use `V = Volfuel(h)`) that gives the volume of fuel in the tank (in gallons) as a function of the height h (measured from the bottom). Use the function to make a plot of the volume as a function of h for $0 \le h \le 70$ in.



8. The surface area S of a ring in shape of a torus with an inner radius r and a diameter d is given by:

$$S = \pi^2(2r + d)d$$

The ring is to be plated with a thin layer of coating. The weight of the coating W can be calculated approximately as $W = \gamma S t$, where γ is the specific weight of the coating material and t is its thickness. Write an anonymous function that calculates the weight of the coating. The function should have four input arguments, r, d, t, and γ. Use the anonymous function to calculate the weight of a gold coating ($\gamma = 0.696$ lb/in.^{3}) of a ring with $r = 0.35$ in., $d = 0.12$ in., and $t = 0.002$ in.



9. The monthly deposit into a savings account *S* needed to reach an investment goal *B* can be calculated by the formula

$$M = S\frac{\frac{r}{1200}}{(1+\frac{r}{1200})^{12N}-1}$$

where *M* is the monthly deposit, *S* is the saving goal, *N* is the number of years, and *r* is the annual interest rate (%). Write a MATLAB user-defined function that calculates the monthly deposit into a savings account. For the function name and arguments use `M = invest(S,r,N)`. The input arguments are `S` (the investment goal), `r` (the annual interest rate, %), and `N` (duration of the savings in years). The output `M` is the amount of the monthly deposit. Use the function to calculate the monthly deposit for a 10-year investment if the investment goal is $25,000 and the annual interest rate is 4.25%.

10. The heat index, *HI* (in degrees F), is an apparent temperature. For temperatures higher than 80°F and humidity higher than 40% it is calculated by:

$$HI = C_1 + C_2T + C_3R + C_4TR + C_5T^2 + C_6R^2 + C_7T^2R + C_8TR^2 + C_9R^2T^2$$

where *T* is temperature in degrees F, *R* is the relative humidity in percent, $C_1 = -42.379$, $C_2 = 2.04901523$, $C_3 = 10.14333127$, $C_4 = -0.22475541$, $C_5 = -6.83783 \times 10^{-3}$, $C_6 = -5.481717 \times 10^{-2}$, $C_7 = 1.22874 \times 10^{-3}$, $C_8 = 8.5282 \times 10^{-4}$, and $C_9 = -1.99 \times 10^{-6}$. Write a user-defined function for calculating *HI* for given *T* and *R*. For the function name and arguments use `HI=HeatIn(T,R)`. The input arguments are `T` in °F and, `R` in %, and the output argument is `HI` in °F (rounded to the nearest integer). Use the function to determine the heat index for the following conditions:
(*a*) $T = 95$ °F, $R = 80$ %.
(*b*) $T = 100$ °F, $R = 100$ % (condition in a sauna).

11. The body fat percentage (*BFP*) of a person can be estimated by the formula

$$BFP = 1.2 \times BMI + 0.23 \times Age - 10.8 \times Gender - 0.54$$

where *BMI* is the body mass index, given by $BMI = 703\frac{W}{H^2}$, in which *W* is the weight in pounds and *H* is the height in inches, *Age* is the person's age, and $Gender = 1$ for a male and $Gender = 0$ for a female.

Write a MATLAB user-defined function that calculates the body fat percentage. For the function name and arguments use `BFP = Body-Fat(w,h,age,gen)`. The input arguments are the weight, height, age, and gender (1 for male, 0 for female), respectively. The output argument is the *BEF* value. Use the function to calculate the body fat percentage of:
a) A 35-years-old, 6 ft 2 in. tall, 220 lb male.
b) A 22-years-old, 5 ft 7 in. tall, 135 lb female.

12. Write a user-defined function that calculates grade point average (GPA) on a scale of 0 to 4, where $A = 4$, $B = 3$, $C = 3$, $D = 1$, and $E = 0$. For the function name and arguments use `av = GPA(g,h)`. The input argument `g` is a vector whose elements are letter grades *A*, *B*, *C*, *D*, or *E* entered as strings. The input argument `h` is a vector with the corresponding credit hours. The output argument `av` is the calculated GPA. Use the function to calculate the GPA for a student with the following record:

Grade	B	A	C	E	A	B	D	B
Credit Hours	3	4	3	4	3	4	3	2

For this case the input arguments are:
`g=['BACEABDB']` and `h=[3 4 3 4 3 4 3 2]`.

13. The factorial $n!$ of a positive number (integer) is defined by $n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$, where $0! = 1$. Write a user-defined function that calculates the factorial $n!$ of a number. For function name and arguments use `y=fact(x)`, where the input argument `x` is the number whose factorial is to be calculated, and the output argument `y` is the value $x!$. The function displays an error message if a negative or non-integer number is entered when the function is called. Use `fact` with the following numbers:
(*a*) 12! (*b*) 0! (*c*) –7! (*d*) 6.7!

14. Write a user-defined MATLAB function that determines the vector connecting two points (*A* and *B*). For the function name and arguments use `V=vector(A,B)`. The input arguments to the function are vectors `A` and `B`, each with the Cartesian coordinates of points *A* and *B*. The output `V` is the vector from point *A* to point *B*. If points *A* and *B* have two coordinates each (they are in the *x y* plane), then `V` is a two-element vector. If points *A* and *B* have three coordinates each (general points in space), then `V` is a three-element vector. Use the function `vector` for determining the following vectors.
(*a*) The vector from point (0.5, 1.8) to point (–3, 16).
(*b*) The vector from point (–8.4, 3.5, –2.2) to point (5, –4.6, 15).

15. Write a user-defined MATLAB function that determines the dot product of two vectors. For the function name and arguments use `D=dotpro(u,v)`. The input arguments to the function are two vectors, which can be two- or three-dimensional. The output `D` is the result (a scalar). Use the function `dotpro` for determining the dot 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. 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 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 (1.2, 3.5) to point (12, 15).
    (*b*) In the direction from point (–10, –4, 2.5) to point (–13, 6, –5).

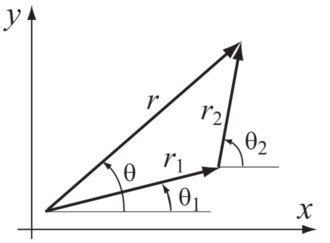
17. Write a user-defined MATLAB function that determines the cross product of two vectors. For the function name and arguments use `w=crosspro(u,v)`. The input arguments to the function are the two vectors, which can be two- or three-dimensional. The output `w` is the result (a vector). Use the function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

18. 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 point *A* to point *B* and ***AC*** is the vector from point *A* to point *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 an other that executes the cross product. (If available, use the user-defined functions from Problems 15 and 17. 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*) $A = (-1.5, -4.2, -3)$, $B = (-5.1, 6.3, 2)$, $C = (12.1, 0, -0.5)$.

19. Write a user-defined function that plots a circle given the coordinates of the center and the radius. For the function name and arguments use `circleplot(x,y,R)`. The input arguments are the *x* and *y* coordinates of the center and the radius. This function has no output arguments. Use the function to plot the following circles:
    (*a*) $x = 3.5$, $y = 2.0$, $R = 8.5$. (*b*) $x = -4.0$, $y = -1.5$, $R = 10$.

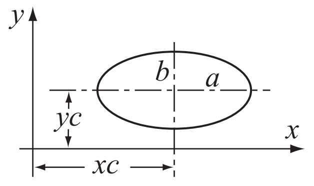
20. Write a user-defined function that plots a circle that passes through three given points. For the function name and arguments use `cirpnts(P)`. The input arguments is a 3×2 matrix in which the two elements of a row are the x and y coordinates of one point. This function has no output arguments. The figure that is created by the function displays the circle and the three points marked with asterisks. Use the function to plot a circle that passes through the points (6, 1.5), (2, 4), (–3, –1.8).

21. 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*) $r_1 = (6, 80°)$, $r_2 = (15, 125°)$.

22. 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 center and the length of the axes. For the function name and arguments use `ellipse-plot(xc,yc,a,b)`. The input arguments `xc` and `yc` are the coordinates of the center, and `a` and `b` are half the lengths of the horizontal and vertical axes (see figure), respectively. This function has no output arguments. Use the function to plot the following ellipses:



(*a*) $xc = 3.5$, $yc = 2.0$, $a = 8.5$, $b = 3$.

(*b*) $xc = -5$, $yc = 1.5$, $a = 4$, $b = 8$.

23. Write a user-defined function that finds all the prime numbers between two numbers m and n. Name the function `pr=prime(m,n)`, where the input arguments `m` and `n` are positive integers, and the output argument `pr` is a vector with the prime numbers. If $m > n$ is entered when the function is called, the error message "The value of n must be larger than the value of m." is displayed. 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. Use the function with:

(*a*) `prime(12,80)` (*b*) `prime(21,63.5)`

(*c*) `prime(100,200)` (*d*) `prime(90,50)`

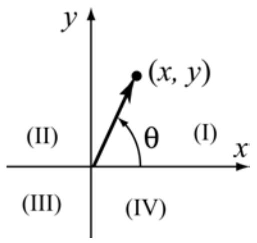
24. The geometric mean *GM* of a set of *n* positive numbers $x_1, x_2, \ldots, x_n$ is defined by:

$$GM = (x_1 \cdot x_2 \cdot \ldots \cdot x_n)^{1/n}$$

Write a user-defined function that calculates the geometric mean of a set of numbers. For function name and arguments use `GM=Geomean(x)`, where the input argument `x` is a vector of numbers (any length) and the output argument `GM` is their geometric mean. The geometric mean is useful for calculating the average return of a stock. The following table gives the returns for IBM stock over the last ten years (a return of 16% means 1.16). Use the user-defined function `Geomean` to calculate the average return of the stock.

Year	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006
Return	1.38	1.76	1.17	0.79	1.42	0.64	1.2	1.06	0.83	1.18

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 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 function `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.

27. 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 matrix and `B` is a matrix of the same size with the elements of `A` rearranged in descending order row after row with the (1,1) element the largest and the (*m*,*n*) element the smallest. If available, use the user-defined function `down-sort` from Problem 26 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.

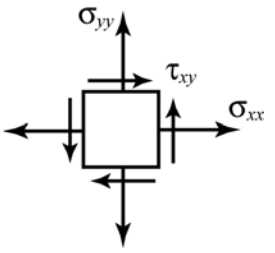
28. Write a user-defined MATLAB function that calculates the determinant of a 3×3 matrix by using the formula:

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

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*) $\begin{bmatrix} -2.5 & 7 & 1 \\ 5 & -3 & -2.6 \\ 4 & 2 & -1 \end{bmatrix}$

29. A two-dimensional state of stress at a point in a loaded material is defined by three components of stress σ_{xx}, σ_{yy}, and τ_{xy}. The maximum and minimum normal stresses (principal stresses) at the point, σ_{max} and σ_{min}, are calculated from the stress components by:



$$\sigma_{\substack{max \\ min}} = \frac{\sigma_{xx} + \sigma_{yy}}{2} \pm \sqrt{\left(\frac{\sigma_{xx} - \sigma_{yy}}{2}\right)^2 + \tau_{xy}^2}$$

Write a user-defined MATLAB function that determines the principal stresses from the stress components. For the function name and arguments use `[Smax,Smin] = princstress(Sxx,Syy,Sxy)`. The input arguments are the three stress components, and the output arguments are the maximum and minimum stresses.

Use the function to determine the principal stresses for the following states of stress:

(*a*) $\sigma_{xx} = -190$ MPa, $\sigma_{yy} = 145$ MPa, and $\tau_{xy} = 110$ MPa.

(*b*) $\sigma_{xx} = 14$ ksi, $\sigma_{yy} = -15$ ksi, and $\tau_{xy} = 8$ ksi.

30. The dew point temperature T_d and the relative humidity *RH* can be calculated (approximately) from the dry-bulb T and wet-bulb T_w temperatures by (http://www.wikipedia.org)

$$e_s = 6.112\exp\left(\frac{17.67T}{T + 243.5}\right) \qquad e_w = 6.112\exp\left(\frac{17.67T_w}{T_w + 243.5}\right)$$

$$e = e_w - p_{sta}(T - T_w)0.00066(1 + 0.00115T_w)$$

$$RH = 100\frac{e}{e_s} \quad T_d = \frac{243.5\ln(e/6.112)}{17.67 - \ln(e/6.112)}$$

where the temperatures are in degrees Celsius, RH is in %, and p_{sta} is the barometric pressure in units of millibars.

Write a user-defined MATLAB function that calculates the dew point temperature and relative humidity for given dry-bulb and wet-bulb temperatures and barometric pressure. For the function name and arguments use `[Td,RH] = DewptRhum(T,Tw,BP)`, where the input arguments are T , T_w and p_{sta}, and the output arguments are T_d and RH. The values of the output arguments should be rounded to the nearest tenth. Use the user-defined function `dewpoint` for calculating the dew point temperature and relative humidity for the following cases:

(*a*) $T = 25$ °C, $T_w = 19$ °C, $p_{sta} = 985$ mbar.

(*b*) $T = 36$ °C, $T_w = 31$ °C, $p_{sta} = 1020$ mbar.

31. Write a user-defined MATLAB function that calculates a student's final grade in a course using the scores from three midterm exams, a final exam, and six homework assignments. The midterms are graded on a scale from 0 to 100 and each accounts for 15% of the course grade. The final exam is graded on a scale from 0 to 100 and accounts for 40% of the course grade. The six homework assignments are each graded on a scale from 0 to 10. The homework assignment with the lowest grade is dropped, and the average of the remaining assignments accounts for 15% of the course grade. In addition, the following adjustment is made when the grade is calculated. If the average grade for the three midterms is higher than the grade for the final exam, then the grade of the final exam is not used and the average grade of the three midterms accounts for 85% of the course grade. The program calculates a course grade that is a number between 0 and 100.

For the function name and arguments use `g = fgrade(R)`. The input argument `R` is a matrix in which the elements in each row are the grades of one student. The first six columns are the homework grades (numbers between 0 and 10), the next three columns are the midterm grades (numbers between 0 and 100), and the last column is the final exam grade (a number between 0 and 100). The output from the function, `g`, is a column vector with the student grades for the course. Each row has the course grade of the student with the grades in the corresponding row of the matrix `R`.

The function can be used to calculate the grades of any number of students. For one student the matrix `R` has one row. Use the function for the following cases:

(*a*) Use the Command Window to calculate the course grade of one student with the following grades: 8, 9, 6, 10, 9, 7, 76, 86, 91, 80.

(b) Write a program in a script file. The program asks the user to enter the students' grades in an array (one student per row). The program then calculates the course grades by using the function `fgrade`. Run the script file in the Command Window to calculate the grades of the following four students:

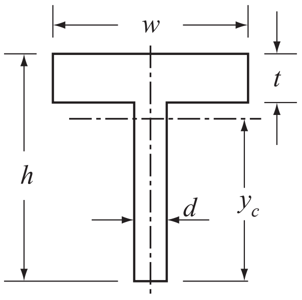
Student A:	7, 10, 6, 9, 10, 9, 91, 71, 81, 88.
Student B:	5, 5, 6, 1, 8, 6, 59, 72, 66, 59.
Student C:	6, 8, 10, 4, 5, 9, 72, 78, 84 78.
Student D:	7, 7, 8, 8, 9, 8, 83, 82, 81 84.

32. In a lottery the player has to select several numbers out of a list. Write a MATLAB program 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.
    (*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.

33. The solution of the nonlinear equation $x^5 - P = 0$ gives the fifth root of the number P. A numerical solution of the equation can be calculated with Newton's method. The solution process starts by choosing a value x_1 as a first estimate of the solution. Using this value, a second, more accurate solution x_2 can be calculated with $x_2 = x_1 - \frac{x_1^5 - P}{5x_1^4}$, which is then used for calculating a third, still more accurate solution x_3, and so on. The general equation for calculating the value of the solution x_{i+1} from the solution x_i is $x_{i+1} = x_i - \frac{x_i^5 - P}{5x_i^4}$. Write a user-defined function that calculates the fifth root of a number. For function name and arguments use `y=fifthroot(P)`, where the input argument `P` is the number whose fifth root is to be determined, and the output argument `y` is the value $\sqrt[5]{P}$. In the program use $x = P$ for the first estimate of the solution. Then, by using the general equation in a loop, calculate new, more accurate solutions. Stop the looping when the estimated relative error E defined by $E = \left|\frac{x_{i+1} - x_i}{x_i}\right|$ is smaller than 0.00001. Use the function `cubic` to calculate:

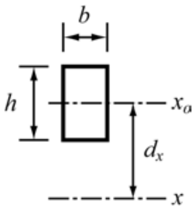
    (*a*) $\sqrt[5]{120}$ (*b*) $\sqrt[5]{16807}$ (*c*) $\sqrt[5]{-15}$

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

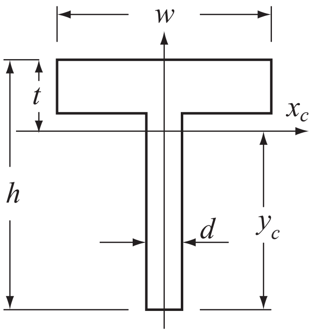


Use the function to determine y_c for an area with w = 240 mm, h = 380 mm, d = 42 mm, and t = 60 mm.

35. 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.



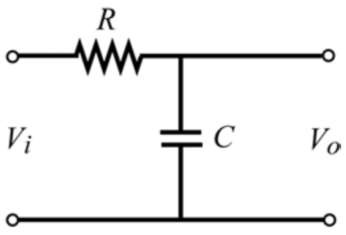
Write a MATLAB user-defined function that determines the area moment of inertia I_{x_c} of a "T" beam about the axis that passes through its centroid (see drawing). For the function name and arguments use `Ixc = IxcTBeam(w,h,t,d)`, where the input arguments `w`, `h`, `t`, and `d` are the dimensions shown in the figure, and the output argument `Ixc` is I_{x_c}. For finding the coordinate y_c of the of the centroid use the user-defined function `centroidT` from Problem 34 as a subfunction inside `IxcTBeam`. (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 of a "T" beam with w = 240 mm, h = 380 mm, d = 42 mm, and t = 60 mm.

36. In a low-pass RC filter (a filter that passes signals with low frequencies), the ratio of the magnitudes of the voltages is given by:

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



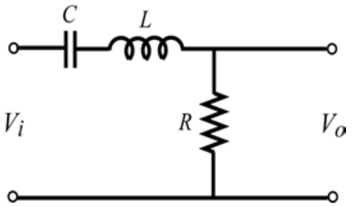
where ω is the frequency of the input signal.

Write a user-defined MATLAB function that calculates the magnitude ratio. For the function name and arguments use `RV = lowpass(R,C,w)`. The input arguments are `R`, the size of the resistor in Ω (ohms); `C`, the size of the capacitor in F (farads); and `w`, the frequency of the input signal in rad/s. Write the function such that `w` can be a vector.

Write a program in a script file that uses the `lowpass` function to generate a plot of *RV* as a function of ω for $10^{-2} \le \omega \le 10^{6}$ rad/s. The plot has a logarithmic scale on the horizontal axis (ω). When the script file is executed, it asks the user to enter the values of *R* and *C*. Label the axes of the plot.

Run the script file with $R = 1200\,\Omega$, and $C = 8\,\mu\text{F}$.

37. A bandpass filter passes signals with frequencies that are within a certain range. In this filter the ratio of the magnitudes of the voltages is given by



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

where ω is the frequency of the input signal.

Write a user-defined MATLAB function that calculates the magnitude ratio. For the function name and arguments use `RV = bandpass(R,C,L,w)`. The input arguments are `R` the size of the resistor in Ω (ohms); `C`, the size of the capacitor in F (farads); `L`, the inductance of the coil in H (henrys); and `w`, the frequency of the input signal in rad/s. Write the function such that `w` can be a vector.

Write a program in a script file that uses the `bandpass` function to generate a plot of *RV* as a function of ω for $10^{-2} \le \omega \le 10^{7}$ rad/s. The plot has a logarithmic scale on the horizontal axis (ω). When the script file is executed, it asks the user to enter the values of *R, L,* and *C*. Label the axes of the plot.

Run the script file for the following two cases:

(*a*) $R = 1100\,\Omega$, $C = 9\ \mu\text{F}$, $L = 7\ \text{mH}$.

(*b*) $R = 500\,\Omega$, $C = 300\ \mu\text{F}$, $L = 400\ \text{mH}$.

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

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

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=Funder(Fun,x0)`, where `Fun` is a name for the function that is passed into `Funder`, and `x0` is the point where the derivative is calculated. Use $h = x_0/10$ in the four-point central difference formula. Use the user-defined function `Funder` to calculate the following:

(*a*) The derivative of $f(x) = x^2e^x$ at $x_0 = 0.25$.

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

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

39. The new coordinates (X_r, Y_r) of a point in the $x\,y$ plane that is rotated about the z axis at an angle θ (positive is clockwise) are given by

$$X_r = X_0\cos\theta - Y_0\sin\theta$$
$$Y_r = X_0\sin\theta + Y_0\cos\theta$$

where (X_0, Y_0) are the coordinates of the point before the rotation. Write a user-defined function that calculates (X_r, Y_r) given (X_0, Y_0) and θ. For function name and arguments use `[xr,yr]=rotation(x,y,q)`, where the input arguments are the initial coordinates and the rotation angle in degrees, and the output arguments are the new coordinates.

(*a*) Use `rotation` to determine the new coordinates of a point originally at $(6.5, 2.1)$ that is rotated about the z-axis by $25°$.

(*b*) Consider the function $y = (x-7)^2 + 1.5$ for $5 \le x \le 9$. Write a program in a script file that makes a plot of the function. Then use `rotation` to rotate all the points that make up the first plot and make a plot of the rotated function. Make both plots in the same figure and set the range of both axes at 0 to 10.