Chapter 6
Programming in MATLAB

A computer program is a sequence of computer commands. In a simple program the commands are executed one after the other in the order they are typed. In this book, for example, all the programs that have been presented so far in script files are simple programs. Many situations, however, require more sophisticated programs in which commands are not necessarily executed in the order they are typed, or different commands (or groups of commands) are executed when the program runs with different input variables. For example, a computer program that calculates the cost of mailing a package uses different mathematical expressions to calculate the cost depending on the weight and size of the package, the content (books are less expensive to mail), and the type of service (airmail, ground, etc.). In other situations there might be a need to repeat a sequence of commands several times within a program. For example, programs that solve equations numerically repeat a sequence of calculations until the error in the answer is smaller than some measure.

MATLAB provides several tools that can be used to control the flow of a program. Conditional statements (Section 6.2) and the `switch` structure (Section 6.3) make it possible to skip commands or to execute specific groups of commands in different situations. `For` loops and `while` loops (Section 6.4) make it possible to repeat a sequence of commands several times.

It is obvious that changing the flow of a program requires some kind of decision-making process within the program. The computer must decide whether to execute the next command or to skip one or more commands and continue at a different line in the program. The program makes these decisions by comparing values of variables. This is done by using relational and logical operators, which are explained in Section 6.1.

It should also be noted that user-defined functions (introduced in Chapter 7) can be used in programming. A user-defined function can be used as a subprogram. When the main program reaches the command line that has the user-defined function, it provides input to the function and “waits” for the results.

The user-defined function carries out the calculations and transfers the results back to the main program, which then continues to the next command.

6.1 Relational and Logical Operators

A relational operator compares two numbers by determining whether a comparison statement (e.g., 5 < 8) is true or false. If the statement is true, it is assigned a value of 1. If the statement is false, it is assigned a value of 0. A logical operator examines true/false statements and produces a result that is true (1) or false (0) according to the specific operator. For example, the logical AND operator gives 1 only if both statements are true. Relational and logical operators can be used in mathematical expressions and, as will be shown in this chapter, are frequently used in combination with other commands to make decisions that control the flow of a computer program.

Relational operators:

Relational operators in MATLAB are:

Relational operator	**Description**
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
~=	Not Equal to

Note that the "equal to" relational operator consists of two = signs (with no space between them), since one = sign is the assignment operator. In other relational operators that consist of two characters, there also is no space between the characters (<=, >=, ~=).

- Relational operators are used as arithmetic operators within a mathematical expression. The result can be used in other mathematical operations, in addressing arrays, and together with other MATLAB commands (e.g., `if`) to control the flow of a program.
- When two numbers are compared, the result is 1 (logical true) if the comparison, according to the relational operator, is true, and 0 (logical false) if the comparison is false.
- If two scalars are compared, the result is a scalar 1 or 0. If two arrays are compared (only arrays of the same size can be compared), the comparison is done *element-by-element*, and the result is a logical array of the same size with 1s and 0s according to the outcome of the comparison at each address.
- If a scalar is compared with an array, the scalar is compared with every element

of the array, and the result is a logical array with 1s and 0s according to the outcome of the comparison of each element.

Some examples are:

```
>> 5>8
```

Checks if 5 is larger than 8.

```
ans =
     0
```

Since the comparison is false (5 is not larger than 8) the answer is 0.

```
>> a=5<10
```

Checks if 5 is smaller than 10, and assigns the answer to a.

```
a =
     1
```

Since the comparison is true (5 is smaller than 10) the number 1 is assigned to a.

Using relational operators in math expression.

```
>> y=(6<10)+(7>8)+(5*3==60/4)
```

Equal to 1 since 6 is smaller than 10.

Equal to 0 since 7 is not larger than 8.

Equal to 1 since 5*3 is equal to 60/4.

```
y =
     2
>> b=[15 6 9 4 11 7 14]; c=[8 20 9 2 19 7 10];
```

Define vectors b and c.

```
>> d=c>=b
```

Checks which c elements are larger than or equal to b elements.

```
d =
     0     1     1     0     1     1     0
```

Assigns 1 where an element of c is larger than or equal to an element of b.

```
>> b == c
```

Checks which b elements are equal to c elements.

```
ans =
     0     0     1     0     0     1     0
>> b~=c
```

Checks which b elements are not equal to c elements.

```
ans =
     1     1     0     1     1     0     1
>> f=b-c>0
```

Subtracts c from b and then checks which elements are larger than zero.

```
f =
     1     0     0     1     0     0     1
>> A=[2 9 4; -3 5 2; 6 7 -1]
```

Define a 3×3 matrix A.

```
A =
     2     9     4
    -3     5     2
     6     7    -1
>> B=A<=2
```

Checks which elements in A are smaller than or equal to 2. Assigns the results to matrix B.

```
B =
     1     0     0
     1     0     1
     0     0     1
```

- The results of a relational operation with vectors, which are vectors with 0s and 1s, are called logical vectors and can be used for addressing vectors. When a logical vector is used for addressing another vector, it extracts from that vector the elements in the positions where the logical vector has 1s. For example:

```
>> r = [8 12 9 4 23 19 10]
```

Define a vector r.

```
r =
     8    12     9     4    23    19    10
>> s=r<=10
```

Checks which r elements are smaller than or equal to 10.

```
s =
     1     0     1     1     0     0     1
```

A logical vector s with 1s at positions where elements of r are smaller than or equal to 10.

```
>> t=r(s)
```

Use s for addresses in vector r to create vector t.

```
t =
     8     9     4    10
```

Vector t consists of elements of r in positions where s has 1s.

```
>> w=r(r<=10)
```

The same procedure can be done in one step.

```
w =
     8     9     4    10
```

- Numerical vectors and arrays with the numbers 0s and 1s are not the same as logical vectors and arrays with 0s and 1s. Numerical vectors and arrays can not be used for addressing. Logical vectors and arrays, however, can be used in arithmetic operations. The first time a logical vector or an array is used in arithmetic operations it is changed to a numerical vector or array.
- Order of precedence: In a mathematical expression that includes relational and arithmetic operations, the arithmetic operations (+, –, *, /, \) have precedence over relational operations. The relational operators themselves have equal precedence and are evaluated from left to right. Parentheses can be used to alter the order of precedence. Examples are:

```
>> 3+4<16/2
```

+ and / are executed first.

```
ans =
     1
```

The answer is 1 since 7 < 8 is true.

```
>> 3+(4<16)/2
```

4 < 16 is executed first, and is equal to 1, since it is true.

```
ans =
    3.5000
```

3.5 is obtained from 3 + 1/2.

Logical operators:

Logical operators in MATLAB are:

Logical operator	Name	Description
& Example: A&B	AND	Operates on two operands (A and B). If both are true, the result is true (1); otherwise the result is false (0).
| Example: A|B	OR	Operates on two operands (A and B). If either one, or both, are true, the result is true (1); otherwise (both are false) the result is false (0).
~ Example: ~A	NOT	Operates on one operand (A). Gives the opposite of the operand; true (1) if the operand is false, and false (0) if the operand is true.

- Logical operators have numbers as operands. A nonzero number is true, and a zero number is false.
- Logical operators (like relational operators) are used as arithmetic operators within a mathematical expression. The result can be used in other mathematical operations, in addressing arrays, and together with other MATLAB commands (e.g., `if`) to control the flow of a program.
- Logical operators (like relational operators) can be used with scalars and arrays.
- The logical operations AND and OR can have both operands as scalars, both as arrays, or one as an array and one as a scalar. If both are scalars, the result is a scalar 0 or 1. If both are arrays, they must be of the same size and the logical operation is done *element-by-element*. The result is an array of the same size with 1s and 0s according to the outcome of the operation at each position. If one operand is a scalar and the other is an array, the logical operation is done between the scalar and each of the elements in the array and the outcome is an array of the same size with 1s and 0s.
- The logical operation NOT has one operand. When it is used with a scalar, the outcome is a scalar 0 or 1. When it is used with an array, the outcome is an array of the same size with 0s in positions where the array has nonzero numbers and 1s in positions where the array has 0s.

Following are some examples:

```
>> 3&7                                                        3 AND 7.
ans =                  3 and 7 are both true (nonzero), so the outcome is 1.
     1
```

```
>> a=5|0
```

5 OR 0 (assign to variable a).

```
a =
     1
```

1 is assigned to a since at least one number is true (nonzero).

```
>> ~25
```

NOT 25.

```
ans =
     0
```

The outcome is 0 since 25 is true (nonzero) and the opposite is false.

```
>> t=25*((12&0)+(~0)+(0|5))
```

Using logical operators in a math expression.

```
t =
    50
>> x=[9 3 0 11 0 15]; y=[2 0 13 -11 0 4];
```

Define two vectors x and y.

```
>> x&y
ans =
     1     0     0     1     0     1
```

The outcome is a vector with 1 in every position where both x and y are true (nonzero elements), and 0s otherwise.

```
>> z=x|y
z =
     1     1     1     1     0     1
```

The outcome is a vector with 1 in every position where either or both x and y are true (nonzero elements), and 0s otherwise.

```
>> ~(x+y)
ans =
     0     0     0     1     1     0
```

The outcome is a vector with 0 in every position where the vector x + y is true (nonzero elements), and 1 in every position where x + y is false (zero elements).

Order of precedence:

Arithmetic, relational, and logical operators can be combined in mathematical expressions. When an expression has such a combination, the result depends on the order in which the operations are carried out. The following is the order used by MATLAB:

Precedence	Operation
1 (highest)	Parentheses (if nested parentheses exist, inner ones have precedence)
2	Exponentiation
3	Logical NOT (~)
4	Multiplication, division
5	Addition, subtraction
6	Relational operators (>, <, >=, <=, ==, ~=)
7	Logical AND (&)
8 (lowest)	Logical OR (|)

If two or more operations have the same precedence, the expression is executed in order from left to right.

It should be pointed out here that the order shown above is the one used since MATLAB 6. Previous versions of MATLAB used a slightly different order (& did not have precedence over |), so the user must be careful. Compatibility problems between different versions of MATLAB can be avoided by using parentheses even when they are not required.

The following are examples of expressions that include arithmetic, relational, and logical operators:

```
>> x=-2; y=5;
```

Define variables x and y.

```
>> -5<x<-1
ans =
     0
```

This inequality is correct mathematically. The answer, however, is false since MATLAB executes from left to right. –5 < *x* is true (=1) and then 1 < –1 is false (0).

```
>> -5<x & x<-1
ans =
     1
```

The mathematically correct statement is obtained by using the logical operator &. The inequalities are executed first. Since both are true (1), the answer is 1.

```
>> ~(y<7)
ans =
     0
```

y < 7 is executed first, it is true (1), and ~1 is 0.

```
>> ~y<7
ans =
     1
```

~*y* is executed first, *y* is true (1) (since y is nonzero), ~1 is 0, and 0 < 7 is true (1).

```
>> ~((y>=8)|(x<-1))
ans =
     0
```

y >= 8 (false), and *x* < –1 (true) are executed first. OR is executed next (true). ~ is executed last, and gives false (0).

```
>> ~(y>=8)|(x<-1)
ans =
     1
```

y >= 8 (false), and *x* < –1 (true) are executed first. NOT of (*y* >= 8) is executed next (true). OR is executed last, and gives true (1).

Built-in logical functions:

MATLAB has built-in functions that are equivalent to the logical operators. These functions are:

`and(A,B)`	equivalent to `A&B`
`or(A,B)`	equivalent to `A|B`
`not(A)`	equivalent to `~A`

In addition, MATLAB has other logical built-in functions, some of which are described in the following table:

Function	Description	Example

`xor(a,b)`	Exclusive or. Returns true (1) if one operand is true and the other is false.	`>> xor(7,0)`
		`ans =`
		`     1`
		`>> xor(7,-5)`
		`ans =`
		`     0`

`all(A)`	Returns 1 (true) if all elements in a vector `A` are true (nonzero). Returns 0 (false) if one or more elements are false (zero).
	If `A` is a matrix, treats columns of `A` as vectors, and returns a vector with 1s and 0s.
		`>>  A=[6  2  15  9  7 11];`
		`>> all(A)`
		`ans =`
		`     1`
		`>>  B=[6  2  15  9  0 11];`
		`>> all(B)`
		`ans =`
		`     0`

`any(A)`	Returns 1 (true) if any element in a vector `A` is true (nonzero). Returns 0 (false) if all elements are false (zero).
	If `A` is a matrix, treats columns of `A` as vectors, and returns a vector with 1s and 0s.
		`>>  A=[6  0  15  0  0 11];`
		`>> any(A)`
		`ans =`
		`     1`
		`>> B = [0 0 0 0 0 0];`
		`>> any(B)`
		`ans =`
		`     0`

`find(A)`
`find(A>d)`
	If `A` is a vector, returns the indices of the nonzero elements.
	If `A` is a vector, returns the address of the elements that are larger than `d` (any relational operator can be used).
		`>> A=[0 9 4 3 7 0 0 1 8];`
		`>> find(A)`
		`ans =`
		`     2     3     4     5     8     9`
		`>> find(A>4)`
		`ans =`
		`     2     5     9`

The operations of the four logical operators, and, or, xor, and not can be summarized in a truth table:

INPUT		OUTPUT				
A	B	AND A&B	OR A|B	XOR (A,B)	NOT ~A	NOT ~B
false	false	false	false	false	true	true
false	true	false	true	true	true	false
true	false	false	true	true	false	true
true	true	true	true	false	false	false

Sample Problem 6-1: Analysis of temperature data

The following were the daily maximum temperatures (in °F) in Washington, DC, during the month of April 2002: 58 73 73 53 50 48 56 73 73 66 69 63 74 82 84 91 93 89 91 80 59 69 56 64 63 66 64 74 63 69 (data from the U.S. National Oceanic and Atmospheric Administration). Use relational and logical operations to determine the following:

(*a*) The number of days the temperature was above 75°.

(*b*) The number of days the temperature was between 65° and 80°.

(*c*) The days of the month when the temperature was between 50° and 60°.

Solution

In the script file below the temperatures are entered in a vector. Relational and logical expressions are then used to analyze the data.

```
T=[58 73 73 53 50 48 56 73 73 66 69 63 74 82 84 ...
   91 93 89 91 80 59 69 56 64 63 66 64 74 63 69];
Tabove75=T>=75;
```

A vector with 1s at addresses where T >= 75.

```
NdaysTabove75=sum(Tabove75)
```

Add all the 1s in the vector Tabove75.

```
Tbetween65and80=(T>=65)&(T<=80);
```

A vector with 1s at addresses where T >= 65 and T <= 80.

```
NdaysTbetween65and80=sum(Tbetween65and80)
```

Add all the 1s in the vector Tbetween65and80.

```
datesTbetween50and60=find((T>=50)&(T<=60))
```

The function find returns the address of the elements in T that have values between 50 and 60.

The script file (saved as Exp6_1) is executed in the Command Window:

```
>> Exp6_1
NdaysTabove75 =                         For 7 days the temp was above 75.
     7
NdaysTbetween65and80 =   For 12 days the temp was between 65 and 80.
    12
datesTbetween50and60 =                        Dates of the month with
     1     4     5     7    21    23          temp between 50 and 60.
```

6.2 Conditional Statements

A conditional statement is a command that allows MATLAB to make a decision of whether to execute a group of commands that follow the conditional statement, or to skip these commands. In a conditional statement, a conditional expression is stated. If the expression is true, a group of commands that follow the statement are executed. If the expression is false, the computer skips the group. The basic form of a conditional statement is:

`if` conditional expression consisting of relational and/or logical operators.

Examples:

```
if a < b
if c >= 5
if a == b
if a ~= 0
if (d<h)&(x>7)
if (x~=13)|(y<0)
```

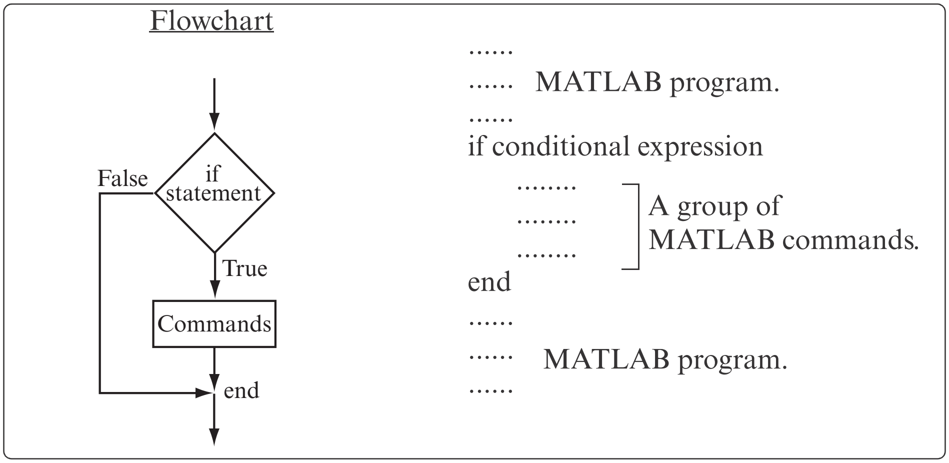
All the variables must have assigned values.

- Conditional statements can be a part of a program written in a script file or a user-defined function (Chapter 7).
- As shown below, for every `if` statement there is an `end` statement.

The `if` statement is commonly used in three structures, `if-end`, `if-else-end`, and `if-elseif-else-end`, which are described next.

6.2.1 *The* `if-end` *Structure*

The `if-end` conditional statement is shown schematically in Figure 6-1. The figure shows how the commands are typed in the program, and a flowchart that symbolically shows the flow, or the sequence, in which the commands are executed. As the program executes, it reaches the `if` statement. If the conditional expression in the `if` statement is true (1), the program continues to execute the



```
......
......  MATLAB program.
......
if conditional expression
      ........  ]  A group of
      ........  ]  MATLAB commands.
      ........  ]
end
......
......  MATLAB program.
......
```

Figure 6-1: The structure of the `if-end` conditional statement.

commands that follow the `if` statement all the way down to the `end` statement. If the conditional expression is false (0), the program skips the group of commands between the `if` and the `end`, and continues with the commands that follow the `end`.

The words `if` and `end` appear on the screen in blue, and the commands between the `if` statement and the `end` statement are automatically indented (they don't have to be), which makes the program easier to read. An example where the `if-end` statement is used in a script file is shown in Sample Problem 6-2.

Sample Problem 6-2: Calculating worker's pay

A worker is paid according to his hourly wage up to 40 hours, and 50% more for overtime. Write a program in a script file that calculates the pay to a worker. The program asks the user to enter the number of hours and the hourly wage. The program then displays the pay.

Solution

The program in a script file is shown below. The program first calculates the pay by multiplying the number of hours by the hourly wage. Then an `if` statement checks whether the number of hours is greater than 40. If so, the next line is executed and the extra pay for the hours above 40 is added. If not, the program skips to the `end`.

```
t=input('Please enter the number of hours worked  ');
h=input('Please enter the hourly wage in $  ');
Pay=t*h;
if t>40
```

```
    Pay=Pay+(t-40)*0.5*h;
end
fprintf('The worker''s pay is  $ %5.2f',Pay)
```

Application of the program (in the Command Window) for two cases is shown below (the file was saved as Workerpay):

```
>> Workerpay
Please enter the number of hours worked  35
Please enter the hourly wage in $  8
The worker’s pay is  $ 280.00
>> Workerpay
Please enter the number of hours worked  50
Please enter the hourly wage in $  10
The worker’s pay is  $ 550.00
```

6.2.2 *The* `if-else-end` *Structure*

The `if-else-end` structure provides a means for choosing one group of commands, out of a possible two groups, for execution. The `if-else-end` structure is shown in Figure 6-2. The figure shows how the commands are typed in the program, and includes a flowchart that illustrates the flow, or the sequence,

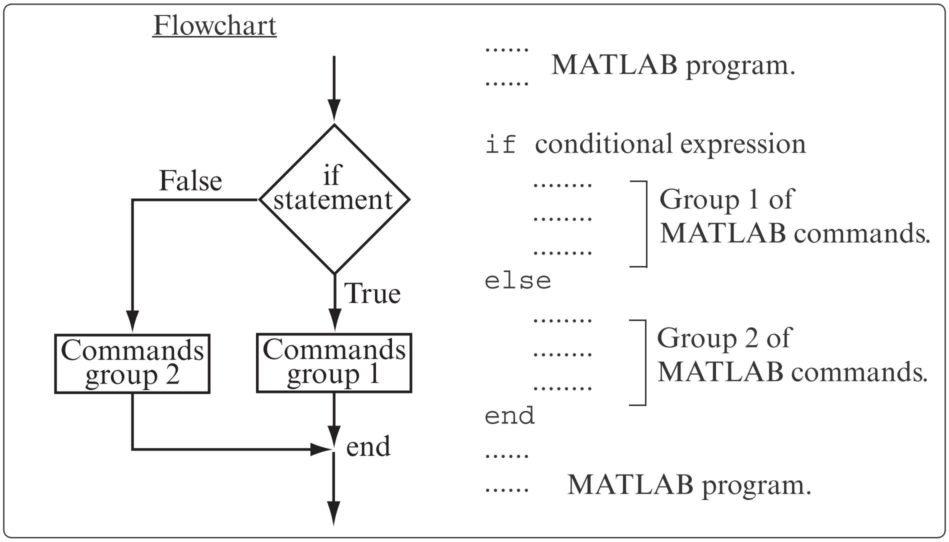


Figure 6-2: The structure of the `if-else-end` **conditional statement.**

in which the commands are executed. The first line is an `if` statement with a conditional expression. If the conditional expression is true, the program executes group 1 of commands between the `if` and the `else` statements and then skips to the `end`. If the conditional expression is false, the program skips to the `else` and then executes group 2 of commands between the `else` and the `end`.

6.2.3 The `if-elseif-else-end` *Structure*

The `if-elseif-else-end` structure is shown in Figure 6-3. The figure shows how the commands are typed in the program, and gives a flowchart that illustrates the flow, or the sequence, in which the commands are executed. This structure includes two conditional statements (`if` and `elseif`) that make it possible to select one out of three groups of commands for execution. The first line is an `if` statement with a conditional expression. If the conditional expression is true, the program executes group 1 of commands between the `if` and the `elseif` statements and then skips to the `end`. If the conditional expression in the `if` statement is false, the program skips to the `elseif` statement. If the conditional expression in the `elseif` statement is true, the program executes group 2 of commands between the `elseif` and the `else` and then skips to the `end`. If the conditional expression in the `elseif` statement is false, the program skips to the `else` and executes group 3 of commands between the `else` and the `end`.

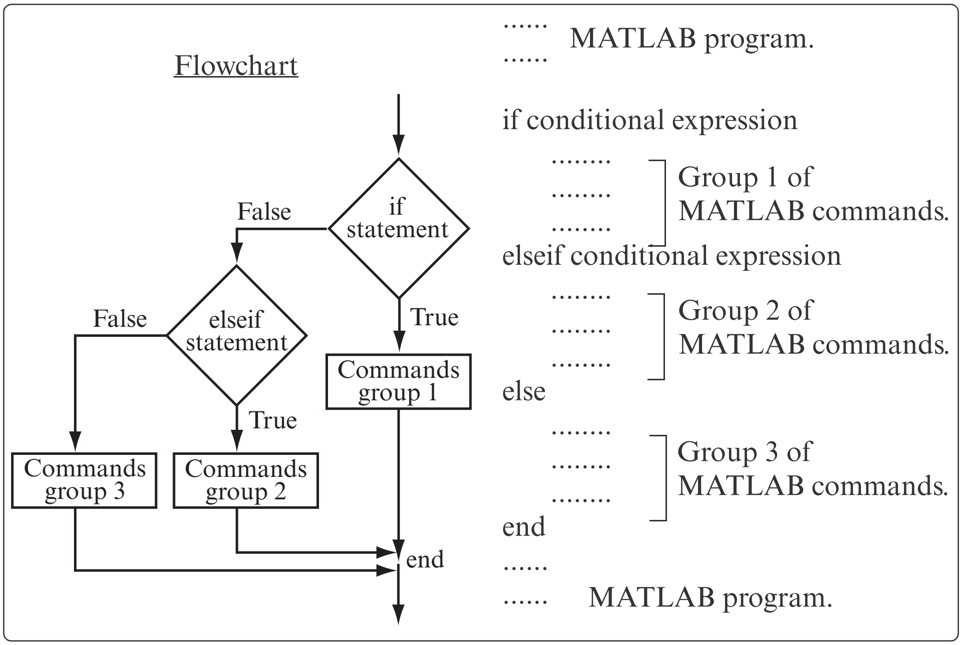
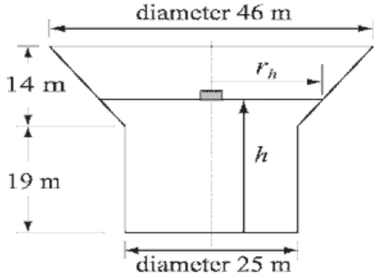


Figure 6-3: The structure of the `if-elseif-else-end` **conditional statement.**

It should be pointed out here that several `elseif` statements and associated groups of commands can be added. In this way more conditions can be included. Also, the `else` statement is optional. This means that in the case of several `elseif` statements and no `else` statement, if any of the conditional statements is true the associated commands are executed; otherwise nothing is executed.

The following example uses the `if-elseif-else-end` structure in a program.

Sample Problem 6-3: Water level in water tower



The tank in a water tower has the geometry shown in the figure (the lower part is a cylinder and the upper part is an inverted frustum of a cone). Inside the tank there is a float that indicates the level of the water. Write a MATLAB program that determines the volume of the water in the tank from the position (height h) of the float. The program asks the user to enter a value of h in m, and as output displays the volume of the water in m^3.

Solution

For $0 \le h \le 19$ m the volume of the water is given by the volume of a cylinder with height h: $V = \pi 12.5^2 h$.

For $19 \le h \le 33$ m the volume of the water is given by adding the volume of a cylinder with $h = 19$ m, and the volume of the water in the cone:

$$V = \pi 12.5^2 \cdot 19 + \frac{1}{3}\pi(h-19)(12.5^2 + 12.5 r_h + r_h^2)$$

where $r_h = 12.5 + \frac{10.5}{14}(h-19)$.

The program is:

```
% The program calculates the volume of the water in the
water tower.
h=input('Please enter the height of the float in meter  ');
if h > 33
   disp('ERROR. The height cannot be larger than 33 m.')
elseif h < 0
   disp('ERROR. The height cannot be a negative number.')
elseif h <= 19
   v = pi*12.5^2*h;
   fprintf('The volume of the water is %7.3f cubic meter.\n',v)
```

```
else
   rh=12.5+10.5*(h-19)/14;
   v=pi*12.5^2*19+pi*(h-19)*(12.5^2+12.5*rh+rh^2)/3;
   fprintf('The volume of the water is %7.3f cubic meter.\n',v)
end
```

The following is the display in the Command Window when the program is used with three different values of water height.

```
Please enter the height of the float in meter  8
The volume of the water is 3926.991 cubic meter.
Please enter the height of the float in meter  25.7
The volume of the water is 14114.742 cubic meter.
Please enter the height of the float in meter  35
ERROR. The height cannot be larger than 33 m.
```

6.3 *THE* switch-case *STATEMENT*

The `switch-case` statement is another method that can be used to direct the flow of a program. It provides a means for choosing one group of commands for execution out of several possible groups. The structure of the statement is shown in Figure 6-4.The first line is the `switch` command, which has the form:

`switch` switch expression

The switch expression can be a scalar or a string. Usually it is a variable that has an assigned scalar or a string. It can also be, however, a mathematical expression that includes pre-assigned variables and can be evaluated.

- Following the `switch` command are one or several `case` commands. Each has a value (can be a scalar or a string) next to it (value1, value2, etc.) and an associated group of commands below it.
- After the last `case` command there is an optional `otherwise` command followed by a group of commands.

The last line must be an `end` statement.

How does the switch-case statement work?

The value of the switch expression in the `switch` command is compared with the values that are next to each of the `case` statements. If a match is found, the group of commands that follow the `case` statement with the match are executed. (Only one group of commands—the one between the `case` that matches and either the `case`, `otherwise`, or `end` statement that is next—is executed).

- If there is more than one match, only the first matching case is executed.

```
......
          MATLAB program.
......

switch switch expression
    case value1
    ........
                   ]  Group 1 of commands.
    ........
    case value2
    ........
                   ]  Group 2 of commands.
    ........
    case value3
    ........
                   ]  Group 3 of commands.
    ........
    otherwise
    ........
                   ]  Group 4 of commands.
    ........
end
......
......    MATLAB program.
```

Figure 6-4: The structure of a `switch-case` statement.

- If no match is found and the `otherwise` statement (which is optional) is present, the group of commands between `otherwise` and `end` is executed.
- If no match is found and the `otherwise` statement is not present, none of the command groups is executed.
- A `case` statement can have more than one value. This is done by typing the values in the form: `{value1, value2, value3, ...}`. (This form, which is not covered in this book, is called a cell array.) The case is executed if at least one of the values matches the value of switch expression.

A Note: In MATLAB only the first matching case is executed. After the group of commands associated with the first matching case are executed, the program skips to the `end` *statement. This is different from the C language, where break statements are required.*

Sample Problem 6-4: Converting units of energy

Write a program in a script file that converts a quantity of energy (work) given in units of either joule, ft-lb, cal, or eV to the equivalent quantity in different units specified by the user. The program asks the user to enter the quantity of energy, its current units, and the desired new units. The output is the quantity of energy in the new units.

The conversion factors are: 1 J = 0.738 ft-lb = 0.239 cal $= 6.24 \times 10^{18}$ eV. Use the program to:

(*a*) Convert 150 J to ft-lb.
(*b*) Convert 2,800 cal to J.
(*c*) Convert 2.7 eV to cal.

Solution

The program includes two sets of switch-case statements and one if-else-end statement. The first switch-case statement is used to convert the input quantity from its initial units to units of joules. The second is used to convert the quantity from joules to the specified new units. The if-else-end statement is used to generate an error message if units are entered incorrectly.

```
Ein=input('Enter the value of the energy (work) to be converted: ');
EinUnits=input('Enter the current units (J, ft-lb, cal, or eV):  ','s');
EoutUnits=input('Enter the new units (J, ft-lb, cal, or eV):  ','s');
error=0;
switch EinUnits
case 'J'
    EJ=Ein;
case 'ft-lb'
    EJ=Ein/0.738;
case 'cal'
    EJ=Ein/0.239;
case 'eV'
    EJ=Ein/6.24e18;
otherwise
    error=1;
end
switch EoutUnits
case 'J'
    Eout=EJ;
case 'ft-lb'
    Eout=EJ*0.738;
case 'cal'
    Eout=EJ*0.239;
case 'eV'
    Eout=EJ*6.24e18;
```

Assign 0 to variable error.

First switch statement. Switch expression is a string with initial units.

Each of the four case statements has a value (string) that corresponds to one of the initial units, and a command that converts Ein to units of J. (Assign the value to EJ.)

Assign 1 to error if no match is found. Possible only if initial units were typed incorrectly.

Second switch statement. Switch expression is a string with new units.

Each of the four case statements has a value (string) that corresponds to one of the new units, and a command that converts EJ to the new units. (Assign the value to Eout.)

```
otherwise
    error=1;
end
if error
    disp('ERROR current or new units are typed incorrectly.')
else
    fprintf('E = %g %s',Eout,EoutUnits)

end
```

Assign 1 to `error` if no match is found. Possible only if new units were typed incorrectly.

`If-else-end` statement.

If `error` is true (nonzero), display an error message.

If `error` is false (zero), display converted energy.

As an example, the script file (saved as EnergyConversion) is used next in the Command Window to make the conversion in part (*b*) of the problem statement.

```
>> EnergyConversion
Enter the value of the energy (work) to be converted: 2800
Enter the current units (J, ft-lb, cal, or eV):  cal
Enter the new units (J, ft-lb, cal, or eV):  J
E = 11715.5 J
```

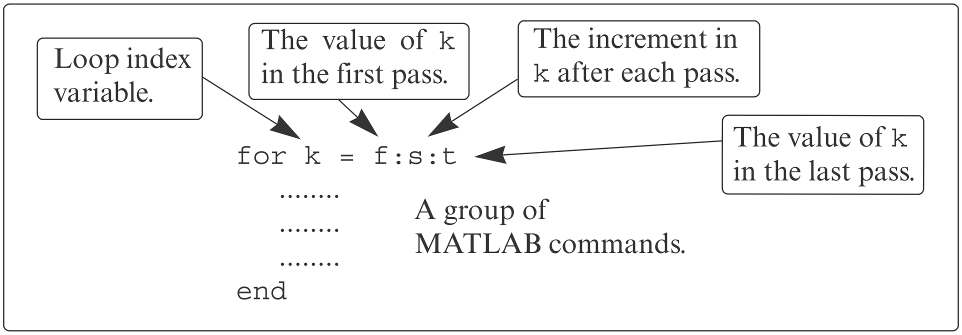
6.4 Loops

A loop is another method to alter the flow of a computer program. In a loop, the execution of a command, or a group of commands, is repeated several times consecutively. Each round of execution is called a pass. In each pass at least one variable, but usually more than one, or even all the variables that are defined within the loop, are assigned new values. MATLAB has two kinds of loops. In `for-end` loops (Section 6.4.1) the number of passes is specified when the loop starts. In `while-end` loops (Section 6.4.2) the number of passes is not known ahead of time, and the looping process continues until a specified condition is satisfied. Both kinds of loops can be terminated at any time with the `break` command (see Section 6.6).

6.4.1 `for-end` *Loops*

In `for-end` loops the execution of a command, or a group of commands, is repeated a predetermined number of times. The form of a loop is shown in Figure 6-5.

- The loop index variable can have any variable name (usually `i`, `j`, `k`, `m`, and `n` are used, but `i` and `j` should not be used if MATLAB is used with complex numbers).



```
for k = f:s:t
    ........
    ........
    ........
end
```

Loop index variable.
The value of k in the first pass.
The increment in k after each pass.
The value of k in the last pass.
A group of MATLAB commands.

Figure 6-5: The structure of a `for-end` loop.

- In the first pass `k = f` and the computer executes the commands between the `for` and `end` commands. Then, the program goes back to the `for` command for the second pass. `k` obtains a new value equal to `k = f + s`, and the commands between the `for` and `end` commands are executed with the new value of `k`. The process repeats itself until the last pass, where `k = t`. Then the program does not go back to the `for`, but continues with the commands that follow the `end` command. For example, if `k` = 1:2:9, there are five passes, and the corresponding values of `k` are 1, 3, 5, 7, and 9.
- The increment `s` can be negative (i.e.; `k` = 25:–5:10 produces four passes with `k` = 25, 20, 15, 10).
- If the increment value `s` is omitted, the value is 1 (default) (i.e.; `k` = 3:7 produces five passes with `k` = 3, 4, 5, 6, 7).
- If `f = t`, the loop is executed once.
- If `f > t` and `s > 0`, or if `f < t` and `s < 0`, the loop is not executed.
- If the values of `k`, `s`, and `t` are such that `k` cannot be equal to `t`, then if `s` is positive, the last pass is the one where `k` has the largest value that is smaller than `t` (i.e., `k` = 8:10:50 produces five passes with `k` = 8, 18, 28, 38, 48). If `s` is negative, the last pass is the one where `k` has the smallest value that is larger than `t`.
- In the `for` command `k` can also be assigned a specific value (typed as a vector). Example: `for k = [7 9 -1 3 3 5].`
- The value of `k` should not be redefined within the loop.
- Each `for` command in a program *must* have an `end` command.
- The value of the loop index variable (`k`) is not displayed automatically. It is possible to display the value in each pass (which is sometimes useful for debugging) by typing `k` as one of the commands in the loop.
- When the loop ends, the loop index variable (`k`) has the value that was last assigned to it.

A simple example of a `for-end` loop (in a script file) is:

```
for k=1:3:10
     x = k^2
end
```

When this program is executed, the loop is executed four times. The value of `k` in the four passes is `k` = 1, 4, 7, and 10, which means that the values that are assigned to `x` in the passes are `x` = 1, 16, 49, and 100, respectively. Since a semicolon is not typed at the end of the second line, the value of `x` is displayed in the Command Window at each pass. When the script file is executed, the display in the Command Window is:

```
>> x =
     1
x =
    16
x =
    49
x =
   100
```

Sample Problem 6-5: Sum of a series

(*a*) Use a `for-end` loop in a script file to calculate the sum of the first *n* terms of the series: $\sum_{k=1}^{n} \frac{(-1)^k k}{2^k}$. Execute the script file for $n = 4$ and $n = 20$.

(*b*) The function sin(*x*) can be written as a Taylor series by:

$$\sin x = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!}$$

Write a user-defined function file that calculates sin(*x*) by using the Taylor series. For the function name and arguments use `y = Tsin(x,n)`. The input arguments are the angle `x` in degrees and `n` the number of terms in the series. Use the function to calculate sin(150°) using three and seven terms.

Solution

(*a*) A script file that calculates the sum of the first *n* terms of the series is shown in the following.

```
n=input('Enter the number of terms ' );
S=0;
for k=1:n
    S=S+(-1)^k*k/2^k;
end
fprintf('The sum of the series is: %f',S)
```

Setting the sum to zero.

for-end loop.

In each pass one element of the series is calculated and is added to the sum of the elements from the previous passes.

The summation is done with a loop. In each pass one term of the series is calculated (in the first pass the first term, in the second pass the second term, and so on) and is added to the sum of the previous elements. The file is saved as Exp6_5a and then executed twice in the Command Window:

```
>> Exp6_5a
Enter the number of terms 4
The sum of the series is: -0.125000
>> Exp7_5a
Enter the number of terms 20
The sum of the series is: -0.222216
```

(*b*) A user-defined function file that calculates sin(*x*) by adding *n* terms of a Taylor series is shown below.

```
function y = Tsin(x,n)
% Tsin calculates the sin using Taylor formula.
% Input arguments:
% x The angle in degrees, n number of terms.

xr=x*pi/180;
y=0;
for k=0:n-1
     y=y+(-1)^k*xr^(2*k+1)/factorial(2*k+1);
end
```

Converting the angle from degrees to radians.

for-end loop.

The first element corresponds to $k = 0$, which means that in order to add *n* terms of the series, in the last loop $k = n - 1$. The function is used in the Command Window to calculate sin(150°) using three and seven terms:

```
>> Tsin(150,3)
ans =
      0.6523
```

Calculating sin(150°) with three terms of Taylor series.

```
>> Tsin(150,7)
ans =
      0.5000
```

Calculating sin(150°) with seven terms of Taylor series.

The exact value is 0.5.

A note about `for-end` loops and element-by-element operations:

In some situations the same end result can be obtained by either using `for-end` loops or using element-by-element operations. Sample Problem 6-5 illustrates how the `for-end` loop works, but the problem can also be solved by using element-by-element operations (see Problems 7 and 8 in Section 3.9). Element-by-element operations with arrays are one of the superior features of MATLAB that provide the means for computing in circumstances that otherwise require loops. In general, element-by-element operations are faster than loops and are recommended when either method can be used.

Sample Problem 6-6: Modify vector elements

A vector is given by V = [5, 17, –3, 8, 0, –7, 12, 15, 20, –6, 6, 4, –7, 16]. Write a program as a script file that doubles the elements that are positive and are divisible by 3 or 5, and, raises to the power of 3 the elements that are negative but greater than –5.

Solution

The problem is solved by using a `for-end` loop that has an `if-elseif-end` conditional statement inside. The number of passes is equal to the number of elements in the vector. In each pass one element is checked by the conditional statement. The element is changed if it satisfies the conditions in the problem statement. A program in a script file that carries out the required operations is:

```
V=[5, 17, -3, 8, 0, -7, 12, 15, 20 -6, 6, 4, -2, 16];
n=length(V);
for k=1:n
   if V(k)>0 & (rem(V(k),3) = = 0 | rem(V(k),5) = = 0)
        V(k)=2*V(k);
   elseif V(k) < 0 & V(k) > -5
        V(k)=V(k)^3;
   end
end
V
```

Setting `n` to be equal to the number of elements in `V`.

`for-end` loop.

`if-elseif-end` statement.

The file is saved as Exp6_6 and then executed in the Command Window:

```
>> Exp6_6
V =
   10   17  -27    8    0   -7   24   30   40   -6   12    4
-8   16
```

6.4.2 `while-end` *Loops*

`while-end` loops are used in situations when looping is needed but the number of passes is not known in advance. In `while-end` loops the number of passes is not specified when the looping process starts. Instead, the looping process continues as long as a stated condition is satisfied. The structure of a `while-end` loop is shown in Figure 6-6.

```
while  conditional expression
    ........
    ........          A group of
    ........          MATLAB commands.
end
```

Figure 6-6: The structure of a `while-end` loop.

The first line is a `while` statement that includes a conditional expression. When the program reaches this line the conditional expression is checked. If it is false (0), MATLAB skips to the `end` statement and continues with the program. If the conditional expression is true (1), MATLAB executes the group of commands that follow between the `while` and `end` commands. Then MATLAB jumps back to the `while` command and checks the conditional expression. This looping process continues until the conditional expression is false.

For a `while-end` loop to execute properly:

- The conditional expression in the `while` command must include at least one variable.
- The variables in the conditional expression must have assigned values when MATLAB executes the `while` command for the first time.
- At least one of the variables in the conditional expression must be assigned a new value in the commands that are between the `while` and the `end`. Otherwise, once the looping starts it will never stop, since the conditional expression will remain true.

An example of a simple `while-end` loop is shown in the following program. In this program a variable `x` with an initial value of 1 is doubled in each pass as long as its value is equal to or smaller than 15.

```
x=1                                         Initial value of x is 1.
while x<=15              The next command is executed only if x <= 15.
    x=2*x                                   In each pass x doubles.
end
```

When this program is executed the display in the Command Window is:

```
x =                                         Initial value of x.
     1
x =
     2
x =                                         In each pass x doubles.
     4
x =
     8
x =                      When x = 16, the conditional expression in the
    16                   while command is false and the looping stops.
```

Important note:

When writing a `while-end` loop, the programmer has to be sure that the variable (or variables) that are in the conditional expression and are assigned new values during the looping process will eventually be assigned values that make the conditional expression in the `while` command false. Otherwise the looping will continue indefinitely (indefinite loop). In the example above if the conditional expression is changed to `x` >= 0.5, the looping will continue indefinitely. Such a situation can be avoided by counting the passes and stopping the looping if the number of passes exceeds some large value. This can be done by adding the maximum number of passes to the conditional expression, or by using the `break` command (Section 6.6).

Since no one is free from making mistakes, a situation of indefinite looping can occur in spite of careful programming. If this happens, the user can stop the execution of an indefinite loop by pressing the **Ctrl + C** or **Ctrl + Break** keys.

Sample Problem 6-7: Taylor series representation of a function

The function $f(x) = e^x$ can be represented in a Taylor series by $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$.

Write a program in a script file that determines e^x by using the Taylor series representation. The program calculates e^x by adding terms of the series and stopping when the absolute value of the term that was added last is smaller than 0.0001. Use a `while-end` loop, but limit the number of passes to 30. If in the

30th pass the value of the term that is added is not smaller than 0.0001, the program stops and displays a message that more than 30 terms are needed.

Use the program to calculate e^2, e^{-4}, and e^{21}.

Solution

The first few terms of the Taylor series are:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

A program that uses the series to calculate the function is shown next. The program asks the user to enter the value of `x`. Then the first term, `an`, is assigned the number 1, and `an` is assigned to the sum `S`. Then, from the second term on, the program uses a `while` loop to calculate the *n*th term of the series and add it to the sum. The program also counts the number of terms `n`. The conditional expression in the `while` command is true as long as the absolute value of the *n*th `an` term is larger than 0.0001, and the number of passes `n` is smaller than 30. This means that if the 30th term is not smaller than 0.0001, the looping stops.

```
x=input('Enter x ' );
n=1; an=1; S=an;
while abs(an) >= 0.0001 & n <= 30            Start of the while loop.
    an=x^n/factorial(n);                     Calculating the nth term.
    S=S+an;                                  Adding the nth term to the sum.
    n=n+1;                                   Counting the number of passes.
end                                          End of the while loop.
if n >= 30                                   if-else-end loop.
    disp('More than 30 terms are needed')
else
fprintf('exp(%f) = %f',x,S)
fprintf('\nThe number of terms used is: %i',n)
end
```

The program uses an `if-else-end` statement to display the results. If the looping stopped because the 30th term is not smaller than 0.0001, it displays a message indicating this. If the value of the function is calculated successfully, it displays the value of the function and the number of terms used. When the program executes, the number of passes depends on the value of `x`. The program (saved as `expox`) is used to calculate e^2, e^{-4}, and e^{21}:

```
>> expox
Enter x 2                                    Calculating exp(2).
```

```
exp(2.000000) = 7.389046
The number of terms used is: 12
>> expox
Enter x -4
exp(-4.000000) = 0.018307
The number of terms used is: 18
>> expox
Enter x 21
More than 30 terms are needed
```

12 terms used.

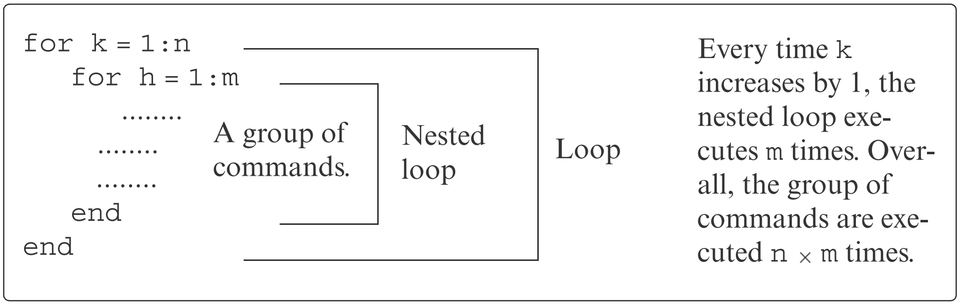
Calculating exp(–4).

18 terms used.

Trying to calculate exp(21).

6.5 Nested Loops and Nested Conditional Statements

Loops and conditional statements can be nested within other loops or conditional statements. This means that a loop and/or a conditional statement can start (and end) within another loop or conditional statement. There is no limit to the number of loops and conditional statements that can be nested. It must be remembered, however, that each `if`, `case`, `for`, and `while` statement must have a corresponding `end` statement. Figure 6-7 shows the structure of a nested `for-end` loop within another `for-end` loop. In the loops shown in this figure, if, for example, `n` = 3 and `m` = 4, then first `k` = 1 and the nested loop executes four times with `h` = 1, 2, 3, 4. Next `k` = 2 and the nested loop executes again four times with `h` = 1, 2, 3, 4. Finally `k` = 3 and the nested loop executes again four times. Every time a nested loop is typed, MATLAB automatically indents the new loop relative to the outside loop. Nested loops and conditional statements are demonstrated in the following sample problem.



```
for k = 1:n
   for h = 1:m
      ........
      ........
      ........
   end
end
```

Every time `k` increases by 1, the nested loop executes `m` times. Overall, the group of commands are executed `n` × `m` times.

Figure 6-7: Structure of nested loops.

Sample Problem 6-8: Creating a matrix with a loop

Write a program in a script file that creates an $n \times m$ matrix with elements that have the following values. The value of each element in the first row is the number of the column. The value of each element in the first column is the number of the row. The rest of the elements each has a value equal to the sum of the element above it and the element to the left. When executed, the program asks the user to enter values for n and m.

Solution

The program, shown below, has two loops (one nested) and a nested `if-elseif-else-end` structure. The elements in the matrix are assigned values row by row. The loop index variable of the first loop, `k`, is the address of the row, and the loop index variable of the second loop, `h`, is the address of the column.

```
n=input('Enter the number of rows ');
m=input('Enter the number of columns ');
A=[];                                  Define an empty matrix A.
for k=1:n                              Start of the first for-end loop.
   for h=1:m                           Start of the second for-end loop.
      if k==1                          Start of the conditional statement.
          A(k,h)=h;                    Assign values to the elements of the first row.
      elseif h==1
          A(k,h)=k;                    Assign values to the elements of the first column.
      else
          A(k,h)=A(k,h-1)+A(k-1,h);    Assign values to other elements.
      end                              end of the if statement.
   end                                 end of the nested for-end loop.
end                                    end of the first for-end loop.
A
```

The program is executed in the Command Window to create a 4 × 5 matrix.

```
>> Chap6_exp8
Enter the number of rows 4
Enter the number of columns 5
```

```
A =
     1     2     3     4     5
     2     4     7    11    16
     3     7    14    25    41
     4    11    25    50    91
```

6.6 *The* `break` *and* `continue` *Commands*

The `break` command:

- When inside a loop (`for` or `while`), the `break` command terminates the execution of the loop (the whole loop, not just the last pass). When the `break` command appears in a loop, MATLAB jumps to the `end` command of the loop and continues with the next command (it does not go back to the `for` command of that loop).
- If the `break` command is inside a nested loop, only the nested loop is terminated.
- When a `break` command appears outside a loop in a script or function file, it terminates the execution of the file.
- The `break` command is usually used within a conditional statement. In loops it provides a method to terminate the looping process if some condition is met — for example, if the number of loops exceeds a predetermined value, or an error in some numerical procedure is smaller than a predetermined value. When typed outside a loop, the `break` command provides a means to terminate the execution of a file, such as when data transferred into a function file is not consistent with what is expected.

The `continue` command:

- The `continue` command can be used inside a loop (`for` or `while`) to stop the present pass and start the next pass in the looping process.
- The `continue` command is usually a part of a conditional statement. When MATLAB reaches the `continue` command, it does not execute the remaining commands in the loop, but skips to the `end` command of the loop and then starts a new pass.

6.7 *Examples of MATLAB Applications*

Sample Problem 6-9: Withdrawing from a retirement account.

A person in retirement is depositing $300,000 in a saving account that pays 5% interest per year. The person plans to withdraw money from the account once a year. He starts by withdrawing $25,000 after the first year, and in future years he increases the amount he withdraws according to the inflation rate. For example, if the inflation rate is 3%, he withdraws $25,750 after the second year. Calculate the number of years the money in the account will last assuming a constant yearly inflation rate of 2%. Make a plot that shows the yearly withdrawals and the balance of the account over the years.

Solution

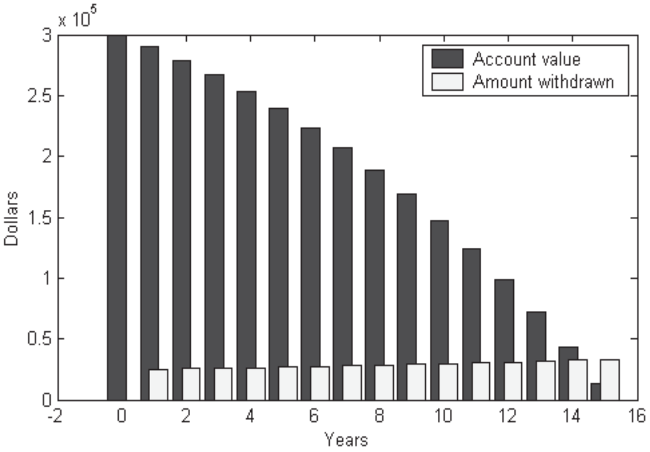
The problem is solved by using a loop (a `while` loop since the number of passes is not known before the loop starts). In each pass the amount to be withdrawn and the account balance are calculated. The looping continues as long as the account balance is larger than or equal to the amount to be withdrawn. The following is a program in a script file that solves the problem. In the program, `year` is a vector in which each element is a year number, `W` is a vector with the amount withdrawn each year, and `AB` is a vector with the account balance each year.

```
rate=0.05; inf=0.02;
clear W AB year
year(1)=0;                                       First element is year 0.
W(1)=0;                                          Initial withdrawal amount.
AB(1)=300000;                                    Initial account balance.
Wnext=25000;                                     The amount to be withdrawn after a year.
ABnext=300000*(1 + rate);                        The account balance after a year.
n=2;
    while ABnext >= Wnext                        while checks if the next balance is
        year(n)=n-1;                             larger than the next withdrawal.
        W(n)=Wnext;                              Amount withdrawn in year n – 1.
        AB(n)=ABnext-W(n);                       Account balance in year n – 1 after withdrawal.
        ABnext=AB(n)*(1+rate);                   The account balance after additional year.
        Wnext=W(n)*(1+inf);
        n=n+1;                                   The amount to be withdrawn
    end                                          after an additional year.
fprintf('The money will last for %f years',year(n-1))
bar(year,[AB' W'],2.0)
```

The program is executed in the following Command Window:

```
>> Chap6_exp9
The money will last for 15 years.
```

The program also generates the following figure (axis labels and legend were added to the plot by using the Plot Editor).



Sample Problem 6-10: Creating a random list

Six singers—John, Mary, Tracy, Mike, Katie, and David—are performing in a competition. Write a MATLAB program that generates a list of a random order in which the singers will perform.

Solution

An integer (1 through 6) is assigned to each name (1 to John, 2 to Mary, 3 to Tracy, 4 to Mike, 5 to Katie, and 6 to David). The program, shown below, first creates a list of the integers 1 through 6 in a random order. The integers are made the elements of six-element vector. This is done by using MATLAB's built-in function `randi` (see Section 3.7) for assigning integers to the elements of the vector. To make sure that all the integers of the elements are different from each other, the integers are assigned one by one. Each integer that is suggested by the `randi` function is compared with all the integers that have been assigned to previous elements. If a match is found, the integer is not assigned, and `randi` is used for suggesting a new integer. Since each singer name is associated with an integer, once the integer list is complete the `switch-case` statement is used to create the corresponding name list.

```
clear, clc
n=6;
```

```
L(1)=randi(n);
for p=2:n
    L(p)=randi(n);
    r=0;
    while r==0
      r=1;
      for k=1:p-1
         if L(k)==L(p)
           L(p)=randi(n);
           r=0;
           break
         end
      end
    end
end
for i=1:n
    switch L(i)
    case 1
        disp('John')
    case 2
        disp('Mary')
    case 3
        disp('Tracy')
    case 4
        disp('Mike')
    case 5
        disp('Katie')
    case 6
        disp('David')
    end
end
```

Assign the first integer to `L(1)`.

Assign the next integer to `L(p)`.

Set `r` to zero.

See explanation below.

Set `r` to 1.

`for` loop compares the integer assigned to `L(p)` to the integers that have been assigned to previous elements.

If a match if found, a new integer is assigned to `L(p)` and `r` is set to zero.

The nested `for` loop is stopped. The program goes back to the `while` loop. Since `r = 0`, the nested loop inside the `while` loop starts again and checks if the new integer that is assigned to `L(p)` is equal to an integer that is already in the vector `L`.

The `switch-case` statement lists the names according to the values of the integers in the elements of `L`.

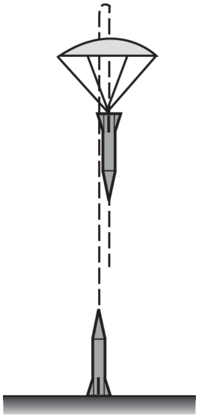
The `while` loop checks that every new integer (element) that is to be added to the vector `L` is not equal any of the integers in elements already in the vector `L`. If a match is found, it keeps generating new integers until the new integer is different from all the integers that are already in `x`.

When the program is executed, the following is displayed in the Command Window. Obviously, a list in a different order will be displayed every time the program is executed.

```
The performing order is:
```

```
Katie
Tracy
David
Mary
John
Mike
```

Sample Problem 6-11: Flight of a model rocket



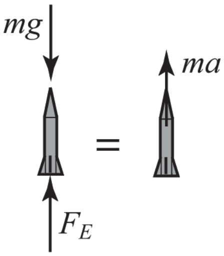
The flight of a model rocket can be modeled as follows. During the first 0.15s the rocket is propelled upward by the rocket engine with a force of 16 N. The rocket then flies up while slowing down under the force of gravity. After it reaches the apex, the rocket starts to fall back down. When its downward velocity reaches 20 m/s, a parachute opens (assumed to open instantly), and the rocket continues to drop at a constant speed of 20 m/s until it hits the ground. Write a program that calculates and plots the speed and altitude of the rocket as a function of time during the flight.

Solution

The rocket is assumed to be a particle that moves along a straight line in the vertical plane. For motion with constant acceleration along a straight line, the velocity and position as a function of time are given by:

$$v(t) = v_0 + at \quad \text{and} \quad s(t) = s_0 + v_0 t + \frac{1}{2}at^2$$

where v_0 and s_0 are the initial velocity and position, respectively. In the computer program the flight of the rocket is divided into three segments. Each segment is calculated in a `while` loop. In every pass the time increases by an increment.



Segment 1: The first 0.15s when the rocket engine is on. During this period, the rocket moves up with a constant acceleration. The acceleration is determined by drawing a free body and a mass acceleration diagram (shown on the right). From Newton's second law, the sum of the forces in the vertical direction is equal to the mass times the acceleration (equilibrium equation):

$$+\uparrow \Sigma F = F_E - mg = ma$$

Solving the equation for the acceleration gives:

$$a = \frac{F_E - mg}{m}$$

The velocity and height as a function of time are:

$$v(t) = 0 + at \quad \text{and} \quad h(t) = 0 + 0 + \frac{1}{2}at^2$$

where the initial velocity and initial position are both zero. In the computer program this segment starts at $t = 0$, and the looping continues as long as $t < 0.15$ s. The time, velocity, and height at the end of this segment are t_1, v_1, and h_1.

<u>Segment 2:</u> The motion from when the engine stops until the parachute opens. In this segment the rocket moves with a constant deceleration g. The speed and height of the rocket as functions of time are given by:

$$v(t) = v_1 - g(t - t_1) \quad \text{and} \quad h(t) = h_1 + v_1(t - t_1) - \frac{1}{2}g(t - t_1)^2$$

In this segment the looping continues until the velocity of the rocket is –20 m/s (negative since the rocket moves down). The time and height at the end of this segment are t_2 and h_2.

<u>Segment 3:</u> The motion from when the parachute opens until the rocket hits the ground. In this segment the rocket moves with constant velocity (zero acceleration). The height as a function of time is given by $h(t) = h_2 - v_{chute}(t - t_2)$, where v_{chute} is the constant velocity after the parachute opens. In this segment the looping continues as long as the height is greater than zero.

A program in a script file that carries out the calculations is shown below.

```
m=0.05;   g=9.81;   tEngine=0.15;   Force=16;   vChute=-20;
Dt=0.01;
clear t v h
n=1;
t(n)=0; v(n)=0; h(n)=0;
% Segment 1
a1=(Force-m*g)/m;
while t(n) < tEngine & n < 50000
    n=n+1;
    t(n)=t(n-1)+Dt;
    v(n)=a1*t(n);
    h(n)=0.5*a1*t(n)^2;
end
v1=v(n); h1=h(n); t1=t(n);
% Segment 2
while v(n) >= vChute & n < 50000
    n=n+1;
    t(n)=t(n-1)+Dt;
```

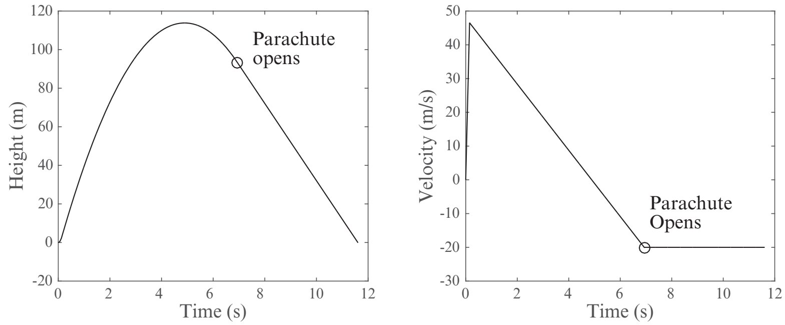
The first `while` loop.

The second `while` loop.

```
    v(n)=v1-g*(t(n)-t1);
    h(n)=h1+v1*(t(n)-t1)-0.5*g*(t(n)-t1)^2;
end
v2=v(n); h2=h(n); t2=t(n);
% Segment 3
while h(n) > 0 & n < 50000
   n=n+1;
    t(n)=t(n-1)+Dt;
    v(n)=vChute;
    h(n)=h2+vChute*(t(n)-t2);
end
subplot(1,2,1)
plot(t,h,t2,h2,'o')
subplot(1,2,2)
plot(t,v,t2,v2,'o')
```

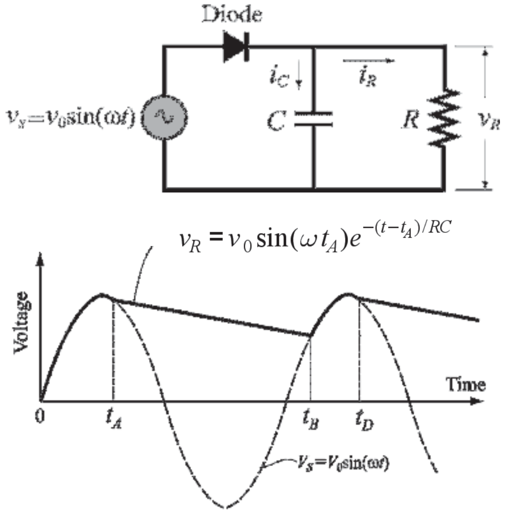
The third `while` loop.

The accuracy of the results depends on the magnitude of the time increment `Dt`. An increment of 0.01 s appears to give good results. The conditional expression in the `while` commands also includes a condition for `n` (if `n` is larger than 50,000 the loop stops). This is done as a precaution to avoid an infinite loop in case there is an error in an of the statements inside the loop. The plots generated by the program are shown below (axis labels and text were added to the plots using the Plot Editor).



Note: The problem can be solved and programmed in different ways. The solution shown here is one option. For example, instead of using `while` loops, the times when the parachute opens and when the rocket hits the ground can be calculated first, and then `for-end` loops can be used instead of the `while` loop. If the times are determined first, it is possible also to use element-by-element calculations instead of loops.

Sample Problem 6-12: AC to DC converter


Diode
i_C
i_R
$v_s = v_0\sin(\omega t)$
C
R
v_R
$v_R = v_0\sin(\omega t_A)e^{-(t-t_A)/RC}$
Voltage
Time
0
t_A
t_B
t_D
$V_s = V_0\sin(\omega t)$

A half-wave diode rectifier is an electrical circuit that converts AC voltage to DC voltage. A rectifier circuit that consists of an AC voltage source, a diode, a capacitor, and a load (resistor) is shown in the figure. The voltage of the source is $v_s = v_0\sin(\omega t)$, where $\omega = 2\pi f$, in which f is the frequency. The operation of the circuit is illustrated in the lower diagram where the dashed line shows the source voltage and the solid line shows the voltage across the resistor. In the first cycle, the diode is on (conducting current) from $t = 0$ until $t = t_A$. At this time the diode turns off and the power to the resistor is supplied by the discharging capacitor. At $t = t_B$ the diode turns on again and continues to conduct current until $t = t_D$. The cycle continues as long as the voltage source is on. In this simplified analysis of this circuit, the diode is assumed to be ideal and the capacitor is assumed to have no charge initially (at $t = 0$). When the diode is on, the resistor's voltage and current are given by:

$$v_R = v_0\sin(\omega t) \quad \text{and} \quad i_R = v_0\sin(\omega t)/R$$

The current in the capacitor is:

$$i_C = \omega C v_0\cos(\omega t)$$

When the diode is off, the voltage across the resistor is given by:

$$v_R = v_0\sin(\omega t_A)e^{-(t-t_A)/RC}$$

The times when the diode switches off (t_A, t_D, and so on) are calculated from the condition $i_R = -i_C$. The diode switches on again when the voltage of the source reaches the voltage across the resistor (time t_B in the figure).

Write a MATLAB program that plots the voltage across the resistor v_R and the voltage of the source v_s as a function of time for $0 \le t \le 70$ ms. The resistance of the load is 1,800 Ω, the voltage source $v_0 = 12$ V, and $f = 60$ Hz. To examine the effect of capacitor size on the voltage across the load, execute the program twice, once with $C = 45$ µF and once with $C = 10$ µF.

Solution

A program that solves the problem is presented below. The program has two parts—one that calculates the voltage v_R when the diode is on, and the other when the diode is off. The `switch` command is used for switching between the two parts. The calculations start with the diode on (the variable `state='on'`), and when $i_R - i_C \leq 0$ the value of `state` is changed to `'off'`, and the program switches to the commands that calculate v_R for this state. These calculations continue until $v_s \geq v_R$, when the program switches back to the equations that are valid when the diode is on.

```
V0=12; C=45e-6; R=1800; f=60;
Tf=70e-3; w=2*pi*f;
clear t VR Vs
t=0:0.05e-3:Tf;
n=length(t);
state='on'
for i=1:n
   Vs(i)=V0*sin(w*t(i));
   switch state
      case 'on'
      VR(i)=Vs(i);
      iR=Vs(i)/R;
      iC=w*C*V0*cos(w*t(i));
      sumI=iR+iC;
      if sumI <= 0
          state='off';
          tA=t(i);
      end
      case 'off'
      VR(i)=V0*sin(w*tA)*exp(-(t(i)-tA)/(R*C));
      if Vs(i) >= VR(i)
         state='on';
      end
   end
end
plot(t,Vs,':',t,VR,'k','linewidth',1)
xlabel('Time (s)'); ylabel('Voltage (V)')
```

Assign `'on'` to the variable `state`.

Calculate the voltage of the source at time t.

Diode is on.

Check if $i_R - i_C \leq 0$.

If true, assign `'off'` to `state`.

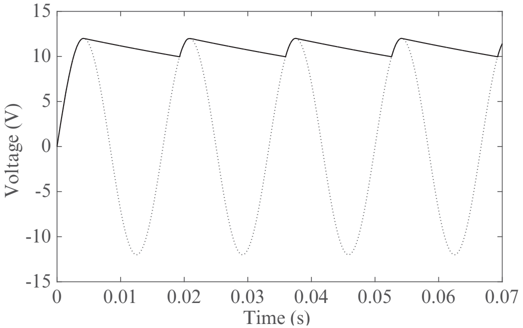
Assign a value to t_A.

Diode is off.

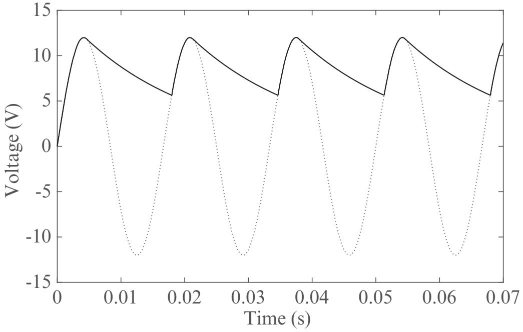
Check if $v_s \geq v_R$.

If true, assign `'on'` to the variable `state`.

The two plots generated by the program are shown below. One plot shows the result with $C = 45$ μF and the other with $C = 10$ μF. It can be observed that with a larger capacitor the DC voltage is smoother (smaller ripple in the wave).



$C = 45$ μF



$C = 10$ μF

6.8 *Problems*

1. Evaluate the following expressions without using MATLAB. Check the answers with MATLAB.

   (*a*) $6 \times 4 > 32 - 3$ (*b*) $y = 4 \times 3 - 7 < 15/3 > -1$

   (*c*) $y = 2 \times (3 < 8/4 + 2)^2 < (-2)^3$ (*d*) $(5 + \sim 0)/3 == 3 - \sim(10/5 - 2)$

2. Given: $d = 6$, $e = 4$, $f = -2$. Evaluate the following expressions without using MATLAB. Check the answers with MATLAB.

   (*a*) $y = d + f >= e > d - e$ (*b*) $y = e > d > f$

   (*c*) $y = e - d <= d - e == f/f$ (*d*) $y = (d/e * f < f) > -1 * (e - d)/f$

3. Given: v = [–2 4 1 0 2 1 2] and w = [2 5 0 1 2 –1 3]. Evaluate the following expressions without using MATLAB. Check the answers with MATLAB.

   (*a*) ~v ==~w (*b*) w >= v

   (*c*) v > ~ –1*w (*d*) v > –1*w

4. Use the vectors *v* and *w* from Problem 3. Use relational operators to create a vector *u* that is made up of the elements of *v* that are smaller than or equal to the elements of *w*.

5. Evaluate the following expressions without using MATLAB. Check the answers with MATLAB.
   (*a*) 0|7&9&–3 (*b*) 7>6&~0<=2
   (*c*) ~4<5|0>=12/6 (*d*) – 7<–5<–2&2+3<=15/3

6. Use loops to create a 4×6 matrix in which the value of each element is two times its row number minus three times its column number. For example, the value of element (2,5) is $2 \times 2 - 3 \times 5 = -11$.

7. Write a program that generates a vector with 30 random integers between –20 and 20 and then finds the sum of all the elements that are divisible by 3.

8. Write a program that asks the user to input a vector of integers of arbitrary length. Then, using a for-end loop the program examines each element of the vector. If the element is positive, its value is doubled. If the element is negative, its value is tripled. The program displays the vector that was entered and the modified vector. Execute the program, and when the program ask the user to input a vector type `randi([-10 20],1,19)`. This creates a 19-element vector with random integers between –10 and 20.

9. Write a program that asks the user to input a vector of integers of arbitrary length. Then, using a for-end loop the program eliminates all the negative elements. The program displays the vector that was entered and the modified vector, and a message that says how many elements were eliminated. Execute the program and when the program ask the user to input a vector type `randi([-15  20],1,25)`. This creates a 25-element vector with random integers between –15 and 20.

10. The daily high temperature (°F) in New York City and Denver, Colorado, during the month of January 2014 is given in the vectors below (data from the U.S. National Oceanic and Atmospheric Administration).
    NYC = [33 33 18 29 40 55 19 22 32 37 58 54 51 52 45 41 45 39 36 45 33 18 19 19 28 34 44 21 23 30 39]
    DEN = [39 48 61 39 14 37 43 38 46 39 55 46 46 39 54 45 52 52 62 45 62 40 25 57 60 57 20 32 50 48 28]
    where the elements in the vectors are in the order of the days in the month. Write a program in a script file that determines and displays the following information:
    (*a*) The average temperature for the month in each city (rounded to the nearest degree).

(*b*) The number of days that the temperature was above the average in each city.

(*c*) The number of days that the temperature in Denver was higher than the temperature in New York.

11. The Pascal triangle can be displayed as elements in a lower-triangular matrix as shown on the right. Write a MATLAB program that creates a $n \times n$ matrix that displays n rows of Pascal's triangle. Use the program to create 4 and 7 rows Pascal's triangles. (One way to calculate the value of the elements in the lower portion of the matrix is $C_{ij} = \frac{(i-1)!}{(j-1)!(i-j)!}$.)

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 \\ 1 & 3 & 3 & 1 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 & 0 \\ 1 & 5 & 10 & 10 & 5 & 1 \end{bmatrix}$$

12. Fibonacci numbers are the numbers in a sequence in which the first three elements are 0, 1, and 1, and the value of each subsequent element is the sum of the previous three elements:

0, 1, 1, 2, 4, 7, 13, 24, ...

Write a MATLAB program in a script file that determines and displays the first 25 Fibonacci numbers.

13. The reciprocal Fibonacci constant ψ is defined by the infinite sum:

$$\psi = \sum_{n=1}^{\infty} \frac{1}{F_n}$$

where F_n are the Fibonacci numbers 1, 1, 2, 3, 5, 8, 13, ... Each element in this sequence of numbers is the sum of the previous two. Start by setting the first two elements equal to 1, then $F_n = F_{n-1} + F_{n-2}$. Write a MATLAB program in a script file that calculates ψ for a given n. Execute the program for $n = 10$, 50, and 100.

14. The value of π can be estimated from:

$$\frac{\pi^3}{32} = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)^3}$$

Write a program (using a loop) that determines π for a given n. Run the program with $n = 10$, $n = 100$, and $n = 1{,}000$. Compare the result with `pi`. (Use `format long`.)

15. The value of π can be estimated from the expression:

$$\frac{2}{\pi} = \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2+\sqrt{2}}}{2} \cdot \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \cdot \ldots$$

Write a MATLAB program in a script file that determine π for any number of terms. The program asks the user to enter the number of terms, and then calculates the corresponding value of π. Execute the program with 5, 10, and 40 terms. Compare the result with `pi`. (Use `format long`.)

16. Write a program that (*a*) generates a vector with 20 random integer elements with integers between 10 and 30, (*b*) replaces all the elements that are not even integers with random integers between 10 and 30, and (*c*) repeats (*b*) until all the elements are even integers. The program should also count how many times (*b*) is repeated before all the elements are even integers. When done, the program displays the vector and a statement that states how many iterations were needed for generating the vector.

17. A vector is given by x = [9 –1.5 13.4 13.3 –2.1 4.6 1.1 5 –6.1 10 0.2]. Using conditional statements and loops, write a program that rearranges the elements of x in order from the smallest to the largest. Do not use MATLAB's built-in function `sort`.

18. The Pythagorean theorem states that $a^2 + b^2 = c^2$. Write a MATLAB program in a script file that finds all the combinations of triples a, b, and c that are positive integers all smaller or equal to 50 that satisfy the Pythagorean theorem. Display the results in a three-column table in which every row corresponds to one triple. The first three rows of the table are:

```
3   4   5
5  12  13
6   8  10
```

19. Write a MATLAB program in a script file that finds and displays all the numbers between 100 and 999 whose product of digits is 6 times the sum of the digits. [e.g. 347 since $3 \times 4 \times 7 = 6(3 + 4 + 7)$]. Use a for-end loop in the program. The loop should start from 100 and end at 999.

20. A safe prime is a prime number that can be written in the form $2p + 1$ where p is also a prime number. For example, 47 is a safe prime since $47 = 2 \times 23 + 1$ and 23 is also a prime number. Write a computer program that finds and displays all the safe primes between 1 and 1,000. Do not use MATLAB's built-in function `isprime`.

21. Sexy primes are two prime numbers that the difference between them is 6. For example, 23 and 29 are sexy primes since $29 - 23 = 6$. Write a computer program that finds all the sexy primes between 1 and 300. The numbers should be displayed in a two-column matrix where each row displays one pair. Do not use MATLAB's built-in function `isprime`.

22. A Mersenne prime is a prime number that is equal to $2^n - 1$, where n is an integer. For example, 31 is a Mersenne prime since $31 = 2^5 - 1$. Write a computer program that finds all the Mersenne primes between 1 and 10,000. Do not use MATLAB's built-in function `isprime`.

23. A perfect number is a positive integer that is equal to the sum of its positive divisors except the number itself. The first two perfect numbers are 6 and 28 since $6 = 1 + 2 + 3$ and $28 = 1 + 2 + 4 + 7 + 14$. Write a computer program that finds the first four perfect numbers.

24. A list of exam scores (S) (in percent out of 100%) is given: 72, 81, 44, 68, 90, 53, 80, 75, 74, 65, 50, 92, 85, 69, 41, 73, 70, 86, 61, 65, 79, 94, 69.
Write a computer program that calculates the average (Av) and standard deviation (Sd) of the scores, which are rounded to the nearest integer. Then, the program determines the letter grade of each of the scores according to the following scheme:

Score (%)	$S \geq Av + 1.3Sd$	$Av + 0.5Sd \leq S < Av + 1.3Sd$
Letter grade	A	B
Score (%)	$Av - 0.5Sd \leq S < Av + 0.5Sd$	$Av - 1.3Sd \leq S < Av - 0.5Sd$
Letter grade	C	D
Score (%)	$S < Av - 1.3Sd$	
Letter grade	F	

The program displays the values of Av and Sd followed by a list that shows the scores and the corresponding letter grade (e.g., 72% Letter grade C).

25. The Taylor series expansion for a^x is:

$$a^x = \sum_{n=0}^{\infty} \frac{(\ln a)^n}{n!} x^n$$

Write a MATLAB program that determines a^x using the Taylor series expansion. The program asks the user to type a value for x. Use a loop for adding the terms of the Taylor series. If c_n is the nth term in the series, then the sum S_n of the n terms is $S_n = S_{n-1} + c_n$. 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 $E < 0.000001$. The program displays the value of a^x. Use the program to calculate:
(a) $2^{3.5}$ (b) $6.3^{1.7}$
Compare the values with those obtained by using a calculator.

26. Write a MATLAB program in a script file that finds a positive integer n such that the sum of all the integers $1 + 2 + 3 + \ldots + n$ is a number between 100 and 1,000 whose three digits are identical. As output, the program displays the integer n and the corresponding sum.

27. The following are formulas for calculating the training heart rate (THR):

$$THR = (MHR - RHR) \times INTEN + RHR$$

where MHR is the maximum heart rate given by (https://en.wikipedia.org/wiki/Heart_rate):

For males: $MHR = \dfrac{203.7}{1 + e^{0.033(age - 104.3)}}$, for females: $MHR = \dfrac{190.2}{1 + e^{0.0453(age - 107.5)}}$,

RHR is the resting heart rate, and $INTEN$ the fitness level (0.55 for low, 0.65 for medium, and 0.8 for high fitness). Write a program in a script file that determines the THR. The program asks users to enter their gender (male or female), age (number), resting heart rate (number), and fitness level (low, medium, or high). The program then displays the training heart rate (rounded to the nearest integer). Use the program for determining the training heart rate for the following two individuals:

(*a*) A 19-year-old male, resting heart rate of 64, and medium fitness level.
(*b*) A 20-year-old female, resting heart rate of 63, and high fitness level.

28. Body mass index (BMI) is a measure of obesity. In standard units, it is calculated by the formula

$$BMI = 703\frac{W}{H^2}$$

where W is weight in pounds, and H is height in inches. The obesity classification is:

BMI	Classification
Below 18.5	Underweight
18.5 to 24.9	Normal
25 to 29.9	Overweight
30 and above	Obese

Write a program in a script file that calculates the BMI of a person. The program asks the person to enter his or her weight (lb) and height (in.). The program displays the result in a sentence that reads: "Your BMI value is XXX, which classifies you as SSSS," where XXX is the BMI value rounded to the nearest tenth, and SSSS is the corresponding classification. Use the program for determining the obesity of the following two individuals:

(*a*) A person 6 ft 2 in. tall with a weight of 180 lb.
(*b*) A person 5 ft 1 in. tall with a weight of 150 lb.

29. Write a program in a script file that calculates the cost of renting a car according to the following price schedule:

Duration of rent	Sedan			SUV		
	Daily rate	Free miles (per day)	Cost of additional mile	Daily rate	Free miles (per day)	Cost of additional mile
1-6 days	\$79	80	\$0.69	\$84	80	\$0.74
7-29 days	\$69	100	\$0.59	\$74	100	\$0.64
30 or more days	\$59	120	\$0.49	\$64	120	\$0.54

The program asks the user to enter the type of car (sedan or SUV), the number of days, and the number of miles driven. The program then displays the cost (rounded to cents) for the rent. Run the program three times for the following cases:
(*a*) Sedan, 10 days, 769 miles. (*b*) SUV, 32 days, 4,056 miles.
(*c*) Sedan, 3 days, 511 miles.

30. Write a program that determines the change given back to a customer in a self-service checkout machine of a supermarket for purchases of up to \$50. The program generates a random number between 0.01 and 50.00 and displays the number as the amount to be paid. The program then asks the user to enter payment, which can be one \$1 bill, one \$5 bill, one \$10 bill, one \$20 bill, or one \$50 bill. If the payment is less than the amount to be paid, an error message is displayed. If the payment is sufficient, the program calculates the change and lists the bills and/or the coins that make up the change, which has to be composed of the least number each of bills and coins. For example, if the amount to be paid is \$2.33 and a \$10 bill is entered as payment, then the change is one \$5 bill, two \$1 bills, two quarters, one dime, one nickel, and two pennies. Execute the program three times.

31. The concentration of a drug in the body C_P can be modeled by the equation:

$$C_P = \frac{D_G}{V_{dP}} \frac{k_a}{k_a - k_{ae}} (e^{-k_e t} - e^{-k_a t})$$

where D_G is the dosage administered (mg), V_d is the volume of distribution (L), k_a is the absorption rate constant (h^{-1}), k_e is the elimination rate constant (h^{-1}), and t is the time (h) since the drug was administered. For a certain drug, the following quantities are given: $D_G = 150\,\text{mg}$, $V_d = 50\,\text{L}$, $k_a = 1.6\,\text{h}^{-1}$, and $k_e = 0.4\,\text{h}^{-1}$.
(*a*) A single dose is administered at $t = 0$. Calculate and plot C_P versus t for 10 h.
(*b*) A first dose is administered at $t = 0$, and subsequently four more doses

are administered at intervals of 4 h (i.e., at $t = 4, 8, 12, 16$). Calculate and plot C_P versus t for 24 h.

32. One numerical method for calculating the cubic root of a number, $\sqrt[3]{P}$ is Halley'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 value x_2 is calculated with $x_2 = x_1(x_1^3 + 2P) / (2x_1^3 + P)$, which is then used for calculating a third, still more accurate value x_3, and so on. The general equation for calculating the value of x_{i+1} from the value of x_i is $x_{i+1} = x_i(x_i^3 + 2P) / (2x_i^3 + P)$. Write a MATLAB program that calculates the cubic root of a number. In the program use $x_1 = P$ for the first estimate of the solution. Then, by using the general equation in a loop, calculate new, more accurate values. 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 program to calculate:

    (*a*) $\sqrt[3]{800}$ (*b*) $\sqrt[3]{59071}$ (*c*) $\sqrt[3]{-31.55}$

33. Write a program in a script file that converts a measure of area given in units of either m², cm², in², ft², yd², or acre to the equivalent quantity in different units specified by the user. The program asks the user to enter a numerical value for the size of an area, its current units, and the desired new units. The output is the size of the area in the new units. Use the program to:

    (*a*) Convert 55 in.² to cm². (*b*) Convert 2400 ft² to m².

    (*c*) Convert 300 cm² to yd².

34. In a one-dimensional random walk, the position x of a walker is computed by:

    $$x_j = x_j + s$$

    where s is a random number. Write a program that calculates the number of steps required for the walker to reach a boundary $x = \pm B$. Use MATLAB's built-in function `randn(1,1)` to calculate s. Run the program 100 times (by using a loop) and calculate the average number of steps when $B = 10$.

35. The Sierpinski triangle can be implemented in MATLAB by plotting points iteratively according to one of the following three rules that are selected randomly with equal probability.

    Rule 1: $x_{n+1} = 0.5x_n$, $y_{n+1} = 0.5y_n$

    Rule 2: $x_{n+1} = 0.5x_n + 0.25$, $y_{n+1} = 0.5y_n + (\sqrt{3})/4$

    Rule 3: $x_{n+1} = 0.5x_n + 0.5$, $y_{n+1} = 0.5y_n$

    Write a program in a script file that calculates the x and y vectors and then plots y versus x as individual points [use `plot(x,y,'^')`]. Start with

$x_1 = 0$ and $y_1 = 0$. Run the program four times with 10, 100, 1,000, and 10,000 iterations.

36. The roots of a cubic equation $a_3x^3 + a_2x^2 + a_1x + a_0 = 0$ can be calculated using the following procedure:
Set: $A = a_2 / a_3$, $B = a_1 / a_3$, and $C = a_0 / a_3$.
Calculate: $D = Q^3 + R^2$,
where $Q = (3B - A^2) / 9$ and $R = (9AB - 27C - 2A^3) / 54$.
If $D > 0$ the equation has complex roots.
If $D = 0$ all roots are real and at least two are equal. The roots are given by:
$x_1 = 2\sqrt[3]{R} - A / 3$, $x_2 = -\sqrt[3]{R} - A / 3$, and $x_3 = -\sqrt[3]{R} - A / 3$.
If $D < 0$ all roots are real and are given by:
$x_1 = 2\sqrt{-Q}\cos(\theta / 3) - A / 3$, $x_2 = 2\sqrt{-Q}\cos(\theta / 3 + 120°) - A / 3$, and
$x_3 = 2\sqrt{-Q}\cos(\theta / 3 + 240°) - A / 3$, where $\cos\theta = R / \sqrt{-Q^3}$.
Write a MATLAB program that determines the real roots of a cubic equation. As input the program asks the user to enter the values of a_3, a_2, a_1, and a_0 as a vector. The program then calculates the value of D. If the equations have complex roots, the message "The equation has complex roots" is displayed. Otherwise the real roots are calculated and displayed. Use the program to solve the following equations:
(*a*) $5x^3 + -34.5x^2 + 36.9x + 8.8 = 0$ (*b*) $2x^3 + -10x^2 + 24x - 15 = 0$
(*c*) $2x^3 + -1.4x^2 - 20.58x + 30.87 = 0$

37. The overall grade in a course is determined from the grades of 10 homework assignments, 2 midterms, and a final exam, using the following scheme:
Homework: Homework assignments are graded on a scale from 0 to 80. The grade of the two lowest assignments is dropped and the average of the eight assignments with the higher grades constitutes 20% of the course grade.
Midterms and final exam: Midterms and final exams are graded on a scale from 0 to 100. If the average of the midterm scores is higher than, or the same as, the score on the final exam, the average of the midterms constitutes 40% of the course grade and the grade of the final exam constitutes 40% of the course grade. If the final exam grade is higher than the average of the midterms, the average of the midterms constitutes 30% of the course grade and the grade of the final exam constitutes 50% of the course grade.

Write a computer program in a script file that determines the course grade for a student. The program first asks the user to enter the 10 homework assignment grades (in a vector), two midterm grades (in a vector), and the grade of the final. Then the program calculates a numerical course grade (a number between 0 and 100). Execute the program for the following cases:
(*a*) Homework assignment grades: 65, 79, 80, 50, 71, 73, 61, 70, 69, 74. Mid-

term grades: 83, 91. Final exam: 84.

(*b*) Homework assignment grades: 70, 69, 83, 45, 90, 89, 52, 78, 100, 87. Midterm grades: 87, 72. Final exam: 90.

38. A Keith number is a number (integer) that appears in a Fibonacci-like sequence that is based on its own decimal digits. For two-decimal digit numbers (10 through 99) a Fibonacci-like sequence is created in which the first element is the tens digit and the second element is the units digit. The value of each subsequent element is the sum of the previous two elements. If the number is a Keith number, then it appears in the sequence. For example, the first two-decimal digit Keith number is 14, since the corresponding Fibonacci-like sequence is 1, 4, 5, 9, 14. Write a MATLAB program that determines and displays all the Keith numbers between 10 and 99.

39. The following MATLAB commands create a sine-shaped signal $y(t)$ that contains random noise:

```
t = 0:.05:10;
y = sin(t)-0.1+0.2*rand(1,length(t));
```

Write a MATLAB program that uses these commands to create a noisy sine-shaped signal. Then the program smooths the signal by using the three-points moving-average method. In this method the value of every point i, except the first and last, is replaced by the average of the value of three adjacent points (i–1, i, and i+1). Make a plot that display the noisy and smoothed signals.