```
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).

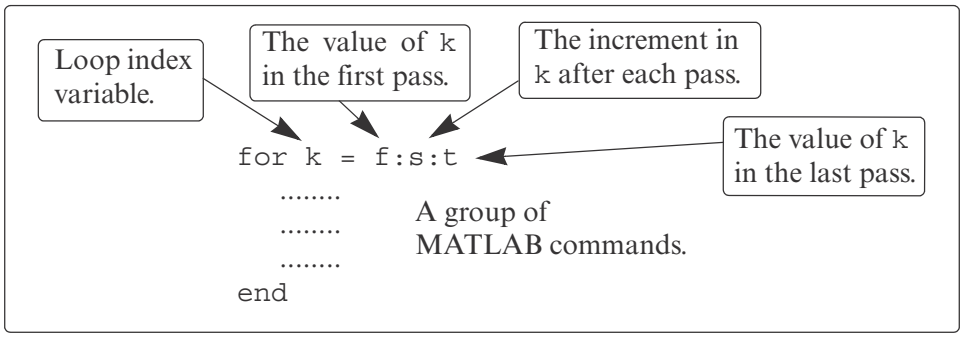

Figure 6-5: The structure of a `for-end` loop.

- 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).
```