```
otherwise                  Assign 1 to error if no match is found. Pos-
    error=1;               sible only if new units were typed incorrectly.
end
if error            If-else-end statement.
    disp('ERROR current or new units are typed incorrectly.')
else                                   If error is true (nonzero),
    fprintf('E = %g %s',Eout,EoutUnits)   display an error message.

end                 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).

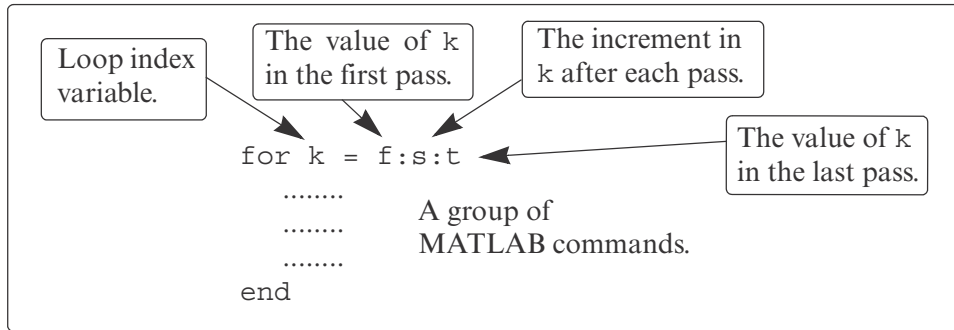Figure 6-5: **The structure of a** `for-end` **loop.**

- 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 semi-colon 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;              Setting the sum to zero.
for k=1:n                               In each pass one element of the
    S=S+(-1)^k*k/2^k;      for-end      series is calculated and is added
                           loop.        to the sum of the elements from
end                                     the previous passes.
fprintf('The sum of the series is: %f',S)
```

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;              Converting the angle from degrees to radians.
y=0;
for k=0:n-1
    y=y+(-1)^k*xr^(2*k+1)/factorial(2*k+1);      for-end
                                                  loop.
end
```

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)      Calculating sin(150°) with three terms of Taylor series.
ans =
      0.6523
```

```
>> Tsin(150,7)          Calculating sin(150°) with seven terms of Taylor series.
ans =
      0.5000                              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);          Setting n to be equal to the number of elements in V.
for k=1:n                                                        for-end
    if V(k)>0 & (rem(V(k),3) = = 0 | rem(V(k),5) = = 0)          loop.
        V(k)=2*V(k);
    elseif V(k) < 0 & V(k) > -5                                  if-
        V(k)=V(k)^3;                                             elseif-
    end                                                          end
end                                                              statement.
V
```