Looping Structures

Leigh J. Little SUNY Brockport September 23, 2022

Contents

1 Introduction

We have seen that it is possible to do more elaborate computation with MATLAB using `if-then` logic. Looping structures will permit even more flexibility in the programs that can be written.

2 Looping operations

Looping operations are used whenever a calculation needs to be performed multiple times.

2.1 `for` loops

Enter the code below into a script named `tloop1.m`.

```
clear
n = input('input n: ');
for i = 1:n
    i
end
disp('End of test')
```

Run the script and input a value of 5 for n. You should see

```
>> tloop1
input n: 5
i =
     1
i =
     2
i =
     3
i =
     4
i =
     5
End of test
```

Run the script for several other positive integer values of n. What do you observe? Based on the test runs, it would seem that the code performs a counting operation.

Now enter the code below into a script called `tloop2.m` and run it for a value of $n = 5$.

```
clear
n = input('input n: ');
for i = 1:2:n
    i
end
disp('End of test')
```

You should see the output

```
>> tloop2
input n: 5
i =
     1
i =
     3
i =
     5
End of test
```

Run the script for several other positive integer values of n. What do you observe in this case? Look carefully at these two scripts. What is the difference in the way the scripts are written?

Finally, run the `tloop1` script for a negative integer value of n. You should see the output below

```
>> tloop1
input n: -2
End of test
>>
```

Notice that in this case, there is no output.

These are examples of `for` loops. These types of loops are used in situations where you know ahead of time how many times the loop must execute. In general, a `for` loop has the syntax

```
for i = startvalue : stepsize : endvalue
   |
   |   Body of Loop
   |
end
```

In the above code segnemt:

- `i` is called the loop index variable (or loop index). This is a variable name that you choose.
- `startvalue` is the starting value of the index `i`.
- `stepsize` is the step increment.
- `endvalue` is the terminal value of `i`.
- Taken as a whole, the `startvalue : stepsize : endvalue` are called the *loop control parameters*.
- The loop control parameters can be variables or static numbers.
- While it is possible for the loop control parameters to be floating point values, this is not recommended for the same type of rounding and storage error issues we discussed at the end of Chapter 4.
- If your `stepsize` is 1 (the most common case), you do not need to include it. This means that

```
for i = 1:10
```

and

```
for i = 1:1:10
```

are the same.

When a loop executes, the flow of the program continues in the following way:

- MATLAB does a test to see if the loop is empty (see next section).
- If the loop is not empty, MATLAB sets the index variable (`i`) to the `startvalue`.
- The statements inside the loop (called the *loop body*) are executed.
- When the end of the loop is reached (the `end` statement) MATLAB increments the value of the index by `stepsize`.
- MATLAB then performs a test:
  - If the new value of the index is greater than `endvalue`, the loop exits. This means that the program continues with the first executable statement past the `end` statement.
  - If the new value of the index is less than or equal to `endvalue`, the loop returns to the first statement in the loop body and executes the statements in the loop body again.

2.2 What About the `n = -2` Example?

When we ran the `tloop1` example for $n = -2$, nothing was output. Why did this happen? Think about how the counting process worked in the previous examples. When MATLAB runs the script for $n = -2$, it is trying to count from 1 to -2 in steps of 1. However, there is no way to do this.

This is an example of an *empty loop*. MATLAB detects that the loop control parameters don't make sense and it skips the loop entirely. More precisely, prior to executing the loop it computes the quantity

```
ntimes = round((endvalue - startvalue + stepsize)/stepsize)
```

The `round` function rounds the input to the nearest integer. If the value of `ntimes` is 0 or negative, the loop is empty and the body is not executed.

As an example, consider when we ran the `tloop2` script for $n = 5$. In this case, the loop control parameters are

```
startvalue = 1
endvalue = 5
stepsize = 2
```

This gives

```
ntimes = round((5-1+2)/2)
       = round(6/2)
       = round(3)
       = 3
```

Thus, this loop will execute the body 3 times. This agrees with what we observed.

Returning to the example with $n = -2$ we have

```
startvalue = 1
endvalue = -2
stepsize = 1
```

This gives

```
ntimes = round((-2 -1 + 1)/1)
       = round(-2)
       = -2
```

This value is negative, so the loop is empty and the body is never executed. This also agrees with what we observed.

2.3 What About Counting Backwards?

Sometimes it is necessary to count backwards. What if we need to count from 5 to 1 and we try

```
clear
for i = 5:1
    i
end
disp('End of test')
```

When you run this, the loop body does not execute. Here, we have

```
startvalue = 5
endvalue = 1
stepsize = 1
```

This gives

```
ntimes = round((1 - 5 + 1)/1)
       = round(-3)
       = -3
```

This value is negative, so the loop is empty and the body is never executed. If you want to step backwards, you need to use a negative stepsize.

```
clear
for i = 5:-1:1
    i
end
disp('End of test')
```

In this case, the output is

```
>> tloop3
i =
    5
i =
    4
i =
    3
i =
    2
i =
    1
End of test
```

2.4 Leave the Loop Index Variable Alone!

Important! You should *never* change the value of the loop index variable yourself. You can use the value of the index variable in calculations, but you should never have a statement like

```
for i =  start_i : step_i : end_i

   i = sin(a/b)      %%     Don't change the value of i yourself.
                     %%     A statement like
                     %%          i = .....
                     %%     should never appear in the body of the loop.
                     %%
                     %%     You can use i in formulas on the right side of an =
                     %%     This is done all the time and is one of the main uses
                     %%     of a loop.

end
```

inside a `for` loop. This leads to unpredictable behavior of your loop and is a terrible idea from the point of view of good programming practice. Unfortunately, MATLAB will allow you to do this. Fortunately, I will not. If I see you doing this in your programs I will mark lots of points off.

3 Accumulation Loops

The activity we did at the beginning of class is one of the most fundamental uses of a loop. This is called an *accumulation process*. The purpose of an accumulation process is to add up a series of values (though sometimes subtraction, multiplication or division of the values is performed instead).

3.1 Mathematical Sums

One attractive feature of the `for` loop structure is that it makes evaluating sums very easy because the mathematical statement of the sum translates almost directly into programming statements. For example, consider the sum

$$\sum_{k=1}^{5} k^2 = 1^2 + 2^2 + 3^2 + 4^2 + 5^2.$$

The limits on the sum are exactly the values we want to use for the starting and ending values of the loop index variable. We can translate this into MATLAB code using

```
total = 0;
for k = 1:5
    total = total + k^2;
end
disp('Sum total')
total
```

This is another example of an accumulation loop and the variable `total` is the *accumulation variable*.

4 Nesting of Loops

Just like with `if-then` statements, you can nest loops inside one another. For example

```
for i = i_start : i_step : i_end
    ....
    ....      Body of i loop
    ....
    for j = j_start : j_step : j_end
        ....
        ....      Body of j Loop
        ....
    end   % end of j loop
    ....
    .... Body of i loop
    ....
end   % end of i loop
```

When nesting loops, the following rules apply:

- The index variables for each loop must be unique.
- The inner loops must logically terminate inside the loop they are contained in.
- Each loop needs its own `end` statement.
- Loops can be nested up to 7 levels deep.

4.1 Nesting Example

An example of how the loop indicies vary in a nested loop is given below:

```
for i = 1:2
    disp('Value of i')
    i
    for j = 1:3
        disp('        Value of j')
        j
    end
end
```

This gives the output

```
>> nest1
Value of i
i =
     1
   Value of j
   j =
       1
   Value of j
   j =
       2
   Value of j
   j =
       3
 Value of i
 i =
     2
     Value of j
     j =
        1
     Value of j
     j =
        2
     Value of j
     j =
        3
```

Notice how the outer loop sets the value of i to 1, then the inner j loop varies from 1 to 3. The j loop terminates, then the i loop sets the value of i to 2 and the inner j loop varies from 1 to 3 again.

5 `while` Loops

We have seen that `for` loops are used when you know how many times the loop must execute. This is not always the case. Situations where the termination of a loop depends on the values currently being computed occur frequently. The `while` loop is useful in these situations. The syntax of a while loop is

```
while (conditional test)
   ....
   ....  Body of while loop
   ....
end
```

The program flow for a `while` loop is as follows:

- When the `while` statement is encountered, MATLAB performs the conditional test.

- If the test is false, the loop is empty and will not execute. Flow continues to the first executable statement past the `end` statement.
- If the test is true, the statements in the body of the loop are executed.
- When the bottom of the loop body is reached, MATLAB returns to the top of the loop and performs the conditional test again. If the test is still true, the loop will execute the body of the loop again. If the test is false, the loop terminates and flow continues to the first executable statement past the `end` statement.
- It is critical that the conditional statement become false at some point, otherwise the loop is infinite. A good example of an infinite loop is

```
lather
rinse
repeat
```

Consider the example below. Enter the statements into a file called `tloop4.m` and execute them

```
clear
i = 0;
while (i < 5)
   i = i + 1
end
```

This gives the output below

```
>> tloop4
i =
     1
i =
     2
i =
     3
i =
     4
i =
     5
```

This is another way to count from 1 to 5.

6 The `break` Statement

Sometimes it is necessary to stop processing a loop immediately. The `break` statement is used to do this. For example

```
for i = 1:10
    i
    if (i > 4)
        break
    end
end
disp('End of test')
```

This will display the output

```
i =
    1
i =
    2
```

```
i =
    3
i =
    4
i =
    5
End of test
```

The `break` statement only applies to the loop that is currently active. For example, if you had

```
for k = 1:3
     k
     for i = 1:10
         i
         if (i > 4)
            break
         end
     end
end
disp('End of test')
```

The `break` in the inner i loop would cause the i loop to terminate, but the k loop would continue to execute. You should run this code segment to see what the output would be.