Loops

Jiseok Chae

Department of Mathematical Sciences
KAIST

Week 5

Contents

Often you will want to execute the same block of commands, possibly with some parameters varying.

Such repititions can be done using loops.

There are mainly two kinds of loops: `while` loops and `for` loops.

The basic template for a `while` loop is:

`while` $\langle P \rangle$
    $\langle$*commands to execute while* $\langle P \rangle$ *is* `TRUE`$\rangle$
`end`

where $\langle P \rangle$ is a conditional statement.

The loop body is executed repeatedly, while $\langle P \rangle$ is `TRUE`.

Let's see an example usage of a `while` loop.

Let's write a function `collatz(n)` which, given an integer `n`, prints out a collatz sequence until it reaches 1.

We assume that the function `collatz_next` we defined before is in the Current Folder.

collatz.m

```
function collatz(n) % no output from the function
disp(n);
while n ~= 1
    n = collatz_next(n); % the function we created
    disp(n);
end
```

An example when $\mathtt{n} = 3$:

```
>> collatz(3)
     3

    10

     5

    16

     8

     4

     2

     1
```

Using `while` loops must be done with special care, as `while` loops are prone to fall into an *infinite loop*, a loop that never ends.

Infinite loops can appear quite unexpectedly.

For example, the `collatz` function does not look like a function that would fall into an infinite loop, as long as the Collatz conjecture is true (as widely believed).

However, if we were to try `collatz(-5)`, note that the Collatz sequence will fall into the loop

$$-5 \rightarrow -14 \rightarrow -7 \rightarrow -20 \rightarrow -10 \rightarrow -5$$

and go on forever without reaching 1, so the function never terminates.

Contents

`while` loops are very versatile, and can be used in almost all (if not all) cases where we need to repeat a block of commands.

However, as we saw just before, there is a caveat.

If you want to run a loop that iterates over a specified set of values, it is better to use `for` loops.

The sentence above might be hard to understand at first glance. Let us see what this means.

The basic template for a `for` loop is:

`for` `i` = ⟨*some vector* v⟩
    ⟨*commands to execute while* `i` *iterates over the elements of* v⟩
`end`

Here, `i` is just a *placeholder* used to denote a parameter. This means that it can be replaced with any other name. For example, the template above is equivalent to

`for` `j` = ⟨*some vector* v⟩
    ⟨*commands to execute while* `j` *iterates over the elements of* v⟩
`end`

Recall the command `a:b` which generates a vector that starts with `a`, is incremented by 1 every element, and contains elements only $\leq$ `b`.

Hence, if we write...

```
for i = 1 : n
    ⟨body of the loop⟩
end
```

...then because `1 : n` is equal to the vector $[1 \ 2 \ \ldots \ n]$, the commands in $\langle$*body of the loop*$\rangle$ will be executed `n` times, and in each iteration the parameter `i` will take the values $1, 2, \ldots, n$ in order.

As a starting example, let us design a function `sqsum(n)` which, given a positive integer `n`, computes the sum $\sum_{i=1}^{n} i^2$.

The observation made in the previous slide suggests the following implementation.

sqsum.m

```
function res = sqsum(n)
res = 0;  % initialize the return variable
for i = 1 : n  % loop for i from 1 to n
    res = res + i * i;  % cumulate i^2
end
```

Use the identity $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ to test the function for yourself.

A slight modification can be used to make a function `sqsum2(a, b)` which, given two positive integers `a` and `b`, computes the sum $\sum_{i=\mathtt{a}}^{\mathtt{b}} i^2$.

sqsum2.m

```
function res = sqsum2(a, b)
res = 0;
for i = a : b  % now the loop is for i from a to b
    res = res + i * i;
end
```

```
>> [sqsum2(4, 8), sqsum(8) - sqsum(3)] % must be equal

ans =

   190   190
```

Since the range of the parameter `i` can be specified by any vector, we can even modify further to construct a function `vecsqsum(v)` which, given a vector $\mathrm{v} \in \mathbb{R}^n$, computes the sum of the square of the elements, $\sum_{i=1}^{n} \mathrm{v}_i^2$.

vecsqsum.m

```
function res = vecsqsum(v)
res = 0;
for i = v  % now the loop is for i in elements of v
    res = res + i * i;
end
```

```
>> vecsqsum([sin(2), cos(3), sin(3), cos(2)])

ans =

    2
```

Thank you!