Octave Tutorial 4: `for` Loops

with extracts from *Introduction to Octave*, by P.J.G. Long

In this tutorial you will learn how to

- write `for` loops in Octave;
- determine whether or not a certain operation in Octave requires a loop.

Suppose we want to perform row operations on a matrix that will result in the entries in the first column of the matrix being one. This can be done by dividing each row by its first entry (we assume that all these entries are nonzero). In other words, given a certain matrix `A` we would want to perform the commands

```
octave#:#> A(1,:)  = A(1,:)/A(1,1);
octave#:#> A(2,:)  = A(2,:)/A(2,1);
⋮
```

and repeating them as many times as the rows in `A`. This is a tedious and time-consuming task, especially when dealing with large matrices. Fortunately Octave allows us to automate the sequence of commands using a `for` loop.

A `for` loop is a programming structure built around some set of statements. The structure always starts with the keyword `for` and ends with the word `end`. A simple example of a `for` loop is the following (this is how it would like if you type the commands into Octave):

```
octave#:#> for i=1:4
> x(i)=i*i
> end
x = 1
x =
   1 4
x =
   1 4 9
x=
   1 4 9 16
```

The final result of this loop is to build a vector `x` whose elements are the squares of the numbers between 1 and 4 in increasing order.

How does this loop work? First of all, the loop generates a variable — in this example `i` — and sets it equal to the first element of the vector `[1:4]` (that is, the vector `[1 2 3 4]`). Then it executes all the commands between the `for` statement and the `end` statement (here there is only one) with that value of `i`. This means that at the first iteration `i=1` and `x(1)=1*1`. Then the program loops back, sets `i` equal to the second element of `[1:4]` and executes the command(s) with this value of `i`, that is, `i=2` and `x(2)=2*2`, and so forth, adding a new element to the vector `x` at each iteration. This iterative process is repeated until `i` has gone through all the elements in `[1:4]`, that is, until `i=4` and `x(4)=4*4`. At that point `i` cannot be incremented further and the loop stops.

Despite the relative simplicity of this `for` loop, it should be noted that if your goal is simply to construct the vector `x=[1 4 9 16]`, using the `for` loop above is not the most efficient way. Octave is optimized for matrix calculations so looping through a vector is slower than performing a matrix operation. For example, it'd be more efficient to build the vector `x` by first defining the vector `x=1:4` and then performing the operation `x=x.^2`. Try it!

So when should a `for` loop be used? `for` loops are best for operations that require looping through, say, rows and columns of a matrix. Do you recall our original task? We started this tutorial by discussing the situation in which we would want to divide each row of a matrix by its first entry. This is a task that can easily be accomplished by looping through the rows of the matrix. Let's see how, but first let's recap what we know about `for` loops. In general, the syntax of a `for` loop is

```
for variable = vector
   statements
end
```

where *variable* is the loop index, *vector* is a vector of some desired length containing the numbers to step through, and *statements* are the commands you'd like Octave to execute at each iteration. Note that the first element of *vector* does not have to be 1, it could be any number, depending on your specific coding needs.

Now suppose you want to write a `for` loop to divide each row of a matrix by its first entry. Let take the matrix `A=[2 2 3; 1 3 1; 3 1 1; 4 5 6]` as an example. Here are a few steps for writing the loop:

• First, choose a name for your loop index. It can be any name as long as you don't give the index the same name as another variable you want to use. For example, let `k` be the index.

• Next, define the range of values which the index should loop through. In this example, the matrix `A` has 4 rows and you need to perform a division on each one of them. This will require four iterations, so `k` should take the values 1 to 4. Thus, the first line of your `for` loop will be `for k = 1:4`.

• Now write the command(s) required to divide each row of `A` by its first entry, keeping in mind that at each iteration the division should be performed on a new row. This means that if at the first iteration the first row `A(1,:)` is divided by its first entry `A(1,1)`, at the second iteration the second row `A(2,:)` should be divided by `A(2,1)`, and so on. Hence, at the k-th iteration the k-th row `A(k,:)` should be divided by `A(k,1)`.
Putting all this together, your `for` loop will look like this,

```
octave#:#> for k = 1:4
> A(k,:)=A(k,:)/A(k,1);
> end
```

Note that we used the semicolon `;` notation to suppress the output of the loop. This is particularly useful when your loop has many iterations.

What about if the matrix `A` has more than 4 rows? How would you modify the loop so that it would work with any matrix? Here is an easy strategy. Begin by determining the number of rows and columns of `A` using the built-in function `size` (you can use the `help` to find out more about this function). Type

```
octave#:#> A=[2 2 3; 1 3 1; 3 1 1; 4 5 6];
octave#:#> [n m] = size(A)
n = 4
m = 3
```

Now `n` is a variable whose value is the number of rows of `A`. Now the commands

```
octave#:#> for k = 1:n
> A(k,:)=A(k,:)/A(k,1);
> end
```

will perform the needed computations. By introducing the new variable `n` as the output of the function `size` you can execute this snippet of code with any size matrix. Just define the matrix and enter the above commands in the correct order. Try it!

Finally, it's worth mentioning that in Octave, as in other languages, it is possible to write a nested loop, that is, a loop within a loop. Look at this example, and before trying it out in Octave, try to explain what this piece of code does.

```
for 1 = 1:5
 for j = 1:4
  A(i,j) = 10*i+j;
 end
end
```

Execrice 1: Without using Octave, explain what sequence of numbers will the following `for` loop print on the screen?

```
n = 10;
for j = 1:n
 n=n-1;
 j
end
```

Exercise 2: Start with a 4 by 4 matrix in which each entry has the value 1 (use the `ones` function). Use a `for` loop to multiply each row i by i (that is, multiply the first row by 1, the second row by 2 etc.). Then use a `for` loop to multiply each column j in the resulting matrix by j (that is, multiply the first column by 1, the second column by 2 etc.).

Exercise 3: Start with `n=5`. Create an n by n matrix `A` with entries that are all 1 except diagonal entries that are set to 5. Use a `for` loop to set the diagonal entries. Try a few different values of n to make sure that your commands work properly.

Exercise 4: Using a `for` loop, write the set of Octave commands required to build a k by j matrix with entries $a_{kj} = \frac{1}{k+i-1}$. Try a few values for k and j.

Exercise 5: Here you will produce the first 25 values of the Fibonacci number sequence. Create a 1 by 25 matrix (a vector) of length 25 and all zeros. Set the 2nd value to 1. Write a `for` loop that loops over the entire vector starting with position 3, and set the value at each iteration to the sum of the previous two values. You must use matrix subscripting (indexing) to retrieve the two previous values. For output, show the initial vector with value 1 at position 2. Also show the final output after the loop has terminated. Suppress all other steps.