Loops

In this tutorial we will demonstrate how the for and the while loop are used. First, the for loop is discussed with examples for row operations on matrices and for Euler's Method to approximate an ODE. Following the for loop , a demonstration of the while loop is given.

We will assume that you know how to create vectors and matrices and know how to index into them. For more information on those topics see one of our tutorials on either vectors (*Introduction to Vectors in Matlab*), matrices (*Introduction to Matrices in Matlab*), or vector operations ((*Vector Functions*).

For Loops

The *for* loop allows us to repeat certain commands. If you want to repeat some action in a predetermined way, you can use the *for* loop. All of the loop structures in matlab are started with a keyword such as *for*, or *while* and they all end with the word *end*. Another deep thought, eh.

The for loop is written around some set of statements, and you must tell Matlab where to start and where to end. Basically, you give a vector in the "for" statement, and Matlab will loop through for each value in the vector:

For example, a simple loop will go around four times each time changing a loop variable, *j*:

```
>> for j=1:4,
   j
end


j =

     1


j =

     2


j =

     3


j =

     4

>>
```

When Matlab reads the *for* statement it constructs a vector, *[1:4]*, and *j* will take on each value within the vector in order. Once Matlab reads the *end* statement, it will execute and repeat the loop. Each time the for statement will update the value of *j* and repeat the statements within the loop. In this example it will print out the value of *j* each time.

For another example, we define a vector and later change the entries. Here we step though and change each individual entry:

```
>> v = [1:3:10]

v =
```

```
     1     4     7    10
```

```
>> for j=1:4,
     v(j) = j;
end
>> v

v =

     1     2     3     4
```

Note, that this is a simple example and is a nice demonstration to show you how a for loop works. However, **DO NOT DO THIS IN PRACTICE!!!!** Matlab is an interpreted language and looping through a vector like this is the slowest possible way to change a vector. The notation used in the first statement is much faster than the loop.

A better example, is one in which we want to perform operations on the rows of a matrix. If you want to start at the second row of a matrix and subtract the previous row of the matrix and then repeat this operation on the following rows, a for loop can do this in short order:

```
>> A = [ [1 2 3]' [3 2 1]' [2 1 3]']

A =

     1     3     2
     2     2     1
     3     1     3

>> B = A;
>> for j=2:3,
    A(j,:) = A(j,:) - A(j-1,:)
end

A =
```

```
     1     3     2
     1    -1    -1
     3     1     3
```

```
A =
```

```
     1     3     2
     1    -1    -1
     2     2     4
```

For a more realistic example, since we can now use loops and perform row operations on a matrix, Gaussian Elimination can be performed using only two loops and one statement:

```
>> for j=2:3,
     for i=j:3,
        B(i,:) = B(i,:) - B(j-1,:)*B(i,j-1)/B(j-1,j-1)
     end
   end
```

```
B =
```

```
     1     3     2
     0    -4    -3
     3     1     3
```

```
B =
```

```
     1     3     2
     0    -4    -3
     0    -8    -3
```

```
B =
```

```
     1     3     2
```

```
     0    -4    -3
     0     0     3
```

Another example where loops come in handy is the approximation of differential equations. The following example approximates the D.E. y'=x^2-y^2, y(0)=1, using Euler's Method. First, the step size, *h*, is defined. Once done, the grid points are found, and an approximation is found. The approximation is simply a vector, *y*, in which the entry *y(j)* is the approximation at *x(j)*.

```
>> h = 0.1;
>> x = [0:h:2];
>> y = 0*x;
>> y(1) = 1;
>> size(x)

ans =

     1    21

>> for i=2:21,
    y(i) = y(i-1) + h*(x(i-1)^2 - y(i-1)^2);
   end
>> plot(x,y)
>> plot(x,y,'go')
>> plot(x,y,'go',x,y)
```

While Loops¶

If you don't like the for loop , you can also use a while loop . The while loop repeats a sequence of commands as long as some condition is met. This can make for a more efficient algorithm. In the previous example the number of time steps to make may be much larger than 20. In such a case the for loop can use up a lot of memory just creating the vector used for the index. A better way of implementing the algorithm is to repeat the same operations but only as long as the number of steps taken is below some threshold. In this example the D.E. y'=x-|y|, y(0)=1, is approximated using Euler's Method:

```
>> h = 0.001;
>> x = [0:h:2];
>> y = 0*x;
>> y(1) = 1;
>> i = 1;
>> size(x)

ans =

           1        2001

>> max(size(x))

ans =

        2001

>> while(i<max(size(x)))
     y(i+1) = y(i) + h*(x(i)-abs(y(i)));
     i = i + 1;
   end
>> plot(x,y,'go')
>> plot(x,y)
```