Loops
(a.k.a. repetition)

Signals & Systems Lab

Lab Notes #3

4.1 The **FOR** loop

sum1ToN.m

```
function runsum = sum1ToN(n)
% sum1ToN returns the sum of integers from 1 to n
% Format of call: sum1ToN(n)

runsum = 0;
for i = 1:n
    runsum = runsum + i;
end
end
```

End of for loop

End of function

```
>> sum1ToN(5)
ans =
    15
```

Straight from the horse's mouth

Syntax

```
for index = values
   program statements
          :
end
```

Description

`for index=values, program statements, end` repeatedly executes one or more MATLAB® statements in a loop. `values` has one of the following forms:

`initval:endval`	increments the `index` variable from `initval` to `endval` by `1`, and repeats execution of `program statements` until `index` is greater than `endval`.
`initval:step:endval`	increments `index` by the value `step` on each iteration, or decrements when `step` is negative.
`valArray`	creates a column vector `index` from subsequent columns of array `valArray` on each iteration. For example, on the first iteration, `index = valArray(:,1)`. The loop executes for a maximum of `n` times, where `n` is the number of columns of `valArray`, given by `numel(valArray, 1, :)`. The input `valArray` can be of any MATLAB data type, including a string, cell array, or struct.

Source: http://www.mathworks.com/help/matlab/ref/for.html

4.1 The **FOR** loop

sum1ToN.m

```
function runsum = sum1ToN(n)
% sum1ToN returns the sum of integers from 1 to n
% Format of call: sum1ToN(n)

runsum = 0;
for i = 1:n
    runsum = runsum + i;
end
end
```

What is the answer for 100?

```
>> sum1ToN(5)
ans =
    15
```

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

<-Gauss

sum1ToN.m

```
function runsum = sum1ToN(n)
% sum1ToN returns the sum of integers from 1 to n
% Format of call: sum1ToN(n)

runsum = 0;
for i = 1:n
    runsum = runsum + i;
end
end
```

PRACTICE 4.1

Write a function *sumMToN* that is similar to the preceding function but will calculate the sum of the integers from *m* to *n*. For example, if the integers 4 and 7 are passed to the function, it will calculate 4 + 5 + 6 + 7:

```
>> sumMToN(4,7)
ans =
    22
```

```
function runfac=factorial(n)
    runfac=1;
    for i=1:n
        runfac=runfac*i;
    end
end

%MATLAB has factorial() built in
```

Vectors and **for** loops go hand-in-hand

myvecsum.m

```
function outarg = myvecsum(vec)
% myvecsum returns the sum of the elements in a
%    vector
% Format of call: myvecsum(vector)

outarg = 0;
for i = 1:length(vec)
    outarg = outarg + vec(i);
end
end
```

```
>> myvecsum([5 9 4])
ans =
    18
```

Vectors and **for** loops go hand-in-hand

```
for i = 1:length(vectorvariable)
    do something with vectorvariable(i)
end
```

Write a script that uses a loop to find the **product** of all values stored in a vector.

```
function product=myvecpro(v)
    product=1;
    for i=1:length(v)
        product=product*v(i);
    end
end

%MATLAB has product() built in
```

Calculate the sum of all the multiples of 3 between 20 and 100, using a for loop.

Calculate the sum of all the multiples of 3 between 20 and 100, using a for loop.

```
sum=0;
for i=21:3:100
        sum=sum+i;
end
disp(sum)
```

Code efficiency - **preallocation**

myveccumsumii.m

```
function outvec = myveccumsumii(vec)
% myveccumsumii imitates cumsum for a vector
% It preallocates the output vector
% Format: myveccumsumii(vector)

outvec = zeros(size(vec));
runsum = 0;
for i = 1:length(vec)
    runsum = runsum + vec(i);
    outvec(i) = runsum;
end
end
```

Nested **for** loops

```
% Prints a box of stars
% How many will be specified by 2 variables
%  for the number of rows and columns

rows = 3;
columns = 5;
% loop over the rows
for i = 1:rows
    % for every row loop to print *'s and then one \n
    for j = 1:columns
        fprintf('*')
    end
    fprintf('\n')
end
```

Running the script displays the output:

```
>> printstars
*****
*****
*****
```

Matrices and **nested for** loops go hand-in-hand

Write a script to print a 10x10 multiplication table

Matrices and **nested for** loops go hand-in-hand

Write a script to print a 10x10 multiplication table

```
for i=1:10
        for j=1:10
                fprintf('%4d',i*j)
        end
        fprintf('\n')
end
```

The **while** loop

factgthigh.m

```
function facgt = factgthigh(high)
% factgthigh returns the first factorial > input
% Format: factgthigh(inputInteger)

i=0;
fac=1;
while fac <= high
    i=i+1;
    fac = fac * i;
end
facgt = fac;
end
```

Note that initializing and incrementing are left up to you when using while.

The **while** loop

It is possible to use more complex conditions for the **while** loop, e.g.

```
while x >= 42 && x <100
```

```
while x >= 42 || ~found
```

Vectorized Code

5.1 Loops w/vectors and matrices

```
outsum = zeros(1,col);
for i = 1:col
    runsum = 0;
    for j = 1:row
        runsum = runsum + mat(j,i);
    end
    outsum(i) = runsum;
end
```

5.2 Direct operations on vectors and matrices

```
>> v  = v * 3
>> v  = v / 2
>> v3 = v1 + v2
>> M  = M + 42
>> M3 = M2 - M1
>> v2 = v1 .* v2
>> M2 = M1 .^2
>> mata = matb ./3
```

Notice the **dot** for *, /, and ^

Achtung! Array operators operate **term-by-term!**

QUIZ: What is printed here?

```
>> M  = [2:3; 4:5]
>> M1 = [4,9; 16,25]
>> M .* M1
>> M .^0.5
```

5.4 Vectorized logical operations

```
>> v = [5 9 3 4 6 11]
>> isgr = vec > 5
isgr =
      0   1   0   0   1   1
>> sum(isgr)
ans =
      3
>> isgr + 5
ans =
      3
```

5.4 Vectorized logical operations

```
>> v = [5 9 3 4 6 11]
>> isgr = vec > 5
isgr =
      0   1   0   0   1   1
>> vec(isgr)
ans =
      9   6   11
```

Other vectorized logical functions

```
>> false(2)
ans =
     0   0
     0   0
>> true(1,5)
ans =
     1   1   1   1   1
```

Other vectorized logical functions

```
>> v = [1 2 0 3 5];
>> any(v)
ans =
     1
>> all(v)
ans =
     0
```

Other vectorized logical functions

```
>> v  = [1 2 0 3 5];
>> v1 = [1 2 0 3 5];
>> find(v<3)
ans =
     1   2   3
>> find(v==3)
ans =
     4
```

find() returns the **index/indices** of the elements that satisfy the condition

```
>> isequal(v,v1)
ans =
     1
```

Other vectorized logical functions

```
>> v1 = [1 2 0 0   5];
>> v2 = [0 3 0 3 42];
>> v1 & v2
ans =
     0   1   0   0   1
>> v1 | v2
ans =
     1   1   0   1   1
```

Lab Work

Sources

Agapie, M. (2013), *CS 344 Class Notes* [used with permission]

Attaway, S. (2012). *MATLAB a practical introduction to programming and problem solving* (2nd ed.). Waltham, MA: Butterworth-Heinemann.