MatLab Control Statements

- MatLab has the usual control statements, such as **for**, **while**, **if-then-else**, **switch/case**, **break** etc. Loops are essential to the use of arrays.

For Loop Examples

- A loop allows a group of statements to be repeated a fixed, predetermined number of times (we'll ignore the prompt symbol >> from now on):

```
for x=lower_bound:step:upper_bound
   group of statements
end % for x
```

- `lower_bound:step:upper_bound` can be replaced by

`lower_bound:upper_bound` if `step` is 1.

- Note that `lower_bound:step:upper_bound` actually defines an array. For example, `2:2:10` is an array with values 2, 4, 6, 8, and 10. A loop:

```
for q=2:2:10
    statements
end % for q
```

would execute `statements` 5 times, with `q` being 2, 4, 6, 8 and 10.

- Loop indices can be 0 in Matlab but array indices cannot be 0 (or negative). Some more examples...

- Forward loop:

```
sum=0;
for m=1:100
    sum=sum+1/m;
 end
>> fprintf('sum=%8.6f\n',sum);
sum=5.187378
```

- Reverse loop:

```
sum=0;
for n=100:-2:0
        sum=sum+1/exp(n);
```

```
    end
fprintf('sum=%8.6f\n',sum);
sum=1.156518
```

- `fprintf` is a print statement that prints `sum` as a floating point number using the format `%8.6f`. This format means 6 digits are printed after the i deciman point and 8 alpha numeric characters can be printed in total. Since there is a decimal point (an alpha numeric character) only 1 digits can be printed to left of the decimal point.

- Use `randperm` to get a list of numbers 1 to 10 in random order. Then a loop can work with these random number as the indices:

```
nums=randperm(10)
```

```
nums =
  6  3  7  8  5  1  2  4  9 10
for n=nums % the loop is executed 10 times,
           % with the values in nums
   x(n)=sin(n*pi/10);
end
format short
x =
 0.3090 0.5878 0.8090 0.9511 1.0000
 0.9511 0.8090 0.5878 0.3090 0.0000
format long
x =
```

```
  Columns 1 through 5
0.309016994374947 0.587785252292473 0.809016994374947
0.951056516295154 1.000000000000000
  Columns 6 through 10
0.951056516295154 0.809016994374947 0.587785252292473
0.309016994374948 0.000000000000000
```

- `for` loops can be nested but an `end` is required for each `for`.

```
for i=1:3
for j=4:7
    fprintf('i=%3d j=%3d i+j=%4d\n',i,j,i+j);
end % for j
```

```
end % for i

i=  1 j=  4 i+j=   5

i=  1 j=  5 i+j=   6

i=  1 j=  6 i+j=   7

i=  1 j=  7 i+j=   8

i=  2 j=  4 i+j=   6

i=  2 j=  5 i+j=   7

i=  2 j=  6 i+j=   8

i=  2 j=  7 i+j=   9

i=  3 j=  4 i+j=   7

i=  3 j=  5 i+j=   8

i=  3 j=  6 i+j=   9
```

```
i=  3 j=  7 i+j=  10
```

- Loops are good for indexing arrays.

```
a(1)=1;
a(2)=2;
a(3)=3;
a(4)=4;
a(5)=5;
a(6)=6;
n=6;
sum=0;
for i=1:n
```

```
  sum=sum+a(i);
end % for i
fprintf('sum=%d n(n+1)/2=%d\n',sum,n*(n+1)/2);
```

prints:

```
sum=21 n(n+1)/2=21
```

That is, $\sum_{i=1}^{n} i$ is equal to $\frac{n(n+1)}{2}$.

- Another nested loop:

```
for n=1:5
for m=1:5
   A(n,m)=n^2+m^2;
```

```
end % m
disp(n) % display or print unformatted n
end % n
     1
     2
     3
     4
     5
A
A =
     2     5    10    17    26
     5     8    13    20    29
```

```
10    13    18    25    34
17    20    25    32    41
26    29    34    41    50
```

While Loop Example

- A while loop can evaluate a group of statements zero to an infinite number of times. The general form of a `while` statement is

```
while expression
   group of statements
end % while
```

- While the booleab expression is true the loop executes. Hopefully, some statement in the body changes the expression to false, otherwise we have an infinite loop.

- One example:

```
% Print all numbers that are powers of 2 below 10000
num = 1; i = 1;
while num < 10000
    fprintf('i=%5d num=%10d\n',i,num);
    i = i+1;
    num = 2^i;
end
```

```
i=    1 num=         1
i=    2 num=         4
i=    3 num=         8
i=    4 num=        16
i=    5 num=        32
```

```
i=     6 num=          64
i=     7 num=         128
i=     8 num=         256
i=     9 num=         512
i=    10 num=        1024
i=    11 num=        2048
i=    12 num=        4096
i=    13 num=        8192
```

- A second example: compute eps, the smallest number that can be added to 1 such that the result is greater than 1, using the finite precision available on a computer. **eps** is called **machine epsilon**. Matlab has builtin constant **eps** that holds this value for your machine.

```
% EPS is used as eps is a built-in MatLab constant
num=0; EPS=1;
while(1+EPS > 1)
   EPS=EPS/2;
   num=num+1;
end % while
% the loop expression becomes false when EPS becomes
% too small. Multiplying it by 2 once give the
% previous EPS value such that EPS+1 != 1
EPS=EPS*2;
num=num-1;
EPS
```

```
EPS =

     2.220446049250313e-16

eps

ans = %

     2.220446049250313e-16

num

num =

    52
```

Double precision is approximately 16 digits so we should expect eps to be near 10^{-16}. 52 is the number of binary digits in the mantissa of a 64 bit floating point number (53 if you count the sign bit). The exponent uses 11 bits.

If-Elseif-Else Statement Example

- The execution of 1 or more commands can be conditionally controlled on the basis of a true/false (boolean) expression. The simplest `if-else-end` construction is:

```
if expression
   group of statements
end % if
```

- When there are 2 alternatives the `if-else-end` construction becomes:

```
if expression
   statements1
```

```
else
   statements2
end % if
```

- When there are 3 or more alternatives the `if-else-end` construction becomes:

```
if expression1
   statements1
elseif expression2
       statements2
...
elseif expression_n_minus_1
```

```
        statements_n_minus_1
else
   statements_n
end

i=6; j=21;
if i > 5
   k=i;
   fprintf('yes\n');
elseif (i>1) & (j==20)
   k=5*i+j;
   fprintf('no\n');
else
```

```
    k=1;
    fprintf('maybe');
end
```

prints:

```
yes
```

Switch-Case Statement Example

- A sequence of statements can conditionally be evaluated on the basis of an equality test using a `switch-case` construction:

```
switch expression
  case test_expression1
       statements1
  case test_expression2
       statements2
  ...
  otherwise statements_n
end
```

- `expression` must either be a boolean (true or false), a character string or a scalar (in the case a character string, the test expressions must also be character strings and equality is tested for). An example in **L04switch.m** is:

```
colour = input('colour=','s');
switch colour
case 'red'
     c=[1 0 0];
case 'green'
     c=[0 1 0];
case 'blue'
     c=[0 0 1]
```

```
otherwise
     error('Invalid choice of colour')
end
```

`str=input(prompt,'s')` returns the entered text as a string, without evaluating the input as an expression. The above example produces the colour tuple for red:

```
colour=red
c =
     1     0     0
>>
```

when `red` is typed after the prompt `colour=`.

- A colour tuple `[x y z]` specifies how much red, green and blue is in a colour. `x`, `y` abd `z` are real numbers in [0,1].
- Red is `[1 0 0]`, green is `[0 1 0]` and blue is `[0 0 1]`.
- Gold is `[1.0.843,0]`, orange is `[1 0.647 0]`, pink is `[1 0.753 0.796]` and brown is `[0.647 0.165 0.165]`.