MATLAB Tutorial

Chapter 2. Programming Structures
2.1. for loops

Programs for numerical simulation often involve repeating a set of commands many times. In MATLAB, we instruct the computer to repeat a block of code by using a for loop. A simple example of a for loop is **for i=1:10** repeats code for i=1,2,...,10
i print out the value of the loop counter **end** This ends the section of code that is repeated.

The counter can be incremented by values other than +1.

```
for i=1:2:10
disp(i);
end
```

This example shows that the counter variables takes on the values 1, 3, 5, 7, 9. After 9, the code next tries i=11, but as 11 is greater than 10 (is not less than or equal to 10) it does not perform the code for this iteration, and instead exits the for loop.

```
for i=10:-1:1
disp(i);
end
```

As the value of the counter integer is changed from one iteration to the next, a common use of for blocks is to perform a given set of operations on different elements of a vector or a matrix. This use of for loops is demonstrated in the example below.

Complex structures can be made by nesting for loops within one another. The nested for loop structure below multiplies an (m x p) matrix with a (p x n) matrix.
A = [1 2 3 4; 11 12 13 14; 21 22 23 24]; A is 3 x 4 matrix
B = [1 2 3; 11 12 13; 21 22 23; 31 32 33]; B is 4 x 3 matrix
im = size(A,1); m is number of rows of A
ip = size(A,2); p is number of columns of A
in = size(B,2); n is number of columns of B
C = zeros(im,in); allocate memory for m x n matrix containing 0's

now we multiply the matrices
for i=1:im iterate over each row of C
for j=1:in iterate over each element in row
for k=1:ip sum over elements to calculate C(i,j)

```
C(i,j) = C(i,j) + A(i,k)*B(k,j);
end
end
end
```

C print out results of code
A*B MATLAB's routine does the same thing

```
clear all
```

2.2. if, case structures and relational operators

In writing programs, we often need to make decisions based on the values of variables in memory. This requires logical operators, for example to discern when two numbers are equal. Common relational operators in MATLAB are : eq(a,b) returns 1 if a is equal to b, otherwise it returns 0

eq(1,2), eq(1,1)
eq(8.7,8.7), eq(8.7,8.71)

When used with vectors or matrices, eq(a,b) returns an array of the same size as a and b with elements of zero where a is not equal b and ones where a equals b. This usage is demonstrated for the examples below.
u = [1 2 3]; w = [4 5 6]; v = [1 2 3]; z = [1 4 3];
eq(u,w), eq(u,v), eq(u,z)
A = [1 2 3; 4 5 6; 7 8 9]; B = [1 4 3; 5 5 6; 7 9 9];
eq(A,B)
this operation can also be called using ==
(1 == 2), (1 == 1), (8.7 == 8.7), (8.7 == 8.71)

ne(a,b) returns 1 if a is not equal to b, otherwise it returns 0
ne(1,2), ne(1,1)
ne(8.7,8.7), ne(8.7,8.71)
ne(u,w), ne(u,v), ne(u,z)
ne(A,B)
another way of calling this operation is to use ~=
(1 ~= 2), (1 ~= 1), (8.7 ~= 8.7), (8.7 ~= 8.71)

lt(a,b) returns 1 if a is less than b, otherwise it returns 0
lt(1,2), lt(2,1), lt(1,1)
lt(8.7,8.71), lt(8.71,8.7), lt(8.7,8.7)
another way of performing this operation is to use <
(1 < 2), (1 < 1), (2 < 1)

le(a,b) returns 1 if a is less than or equal to b, otherwise 0
le(1,2), le(2,1), le(1,1)
le(8.7,8.71), le(8.71,8.7), le(8.7,8.7)
this operation is also performed using <=
(1 <= 1), (1 <= 2), (2 <= 1)

gt(a,b) returns 1 if a is greater than b, otherwise 0
gt(1,2), gt(2,1), gt(1,1)
gt(8.7,8.71), gt(8.71,8.7), gt(8.7,8.7)
this operation is also performed using >
(1 > 2), (1 > 1), (2 > 1)

ge(a,b) returns 1 if a is greater than or equal to b, otherwise 0
ge(1,2), ge(2,1), ge(1,1)
ge(8.7,8.71), ge(8.71,8.7), ge(8.7,8.7)
this operation is also performed using >=
(1 >= 1), (1 >= 2), (2 >= 1)

These operations can be combined to perform more complex logical tests.

(logic1)&(logic2) returns 0 unless both logic1 and logic2 are not equal to zero
((1==1)&(8.7==8.7))
((1==2)&(8.7==8.7))
((1>2)&(8.71>8.7))
((1<2)&(8.7<8.71))
((1>2)&(8.7>8.71))
i1 = 1; i2 = 0; i3=-1;
(i1 & i1), (i1 & i2), (i2 & i1), (i2 & i2), (i1 & i3)
((1==1)&(8.7==8.7)&(1<2))
((1==1)&(8.7==8.7)&(1>2))

This operation can be extended to multiple operations more easily by using the command all(vector1), that returns 1 if all of the elements of vector1 are nonzero, otherwise it returns 0
all([i1 i2 i3]), all([i1 i1 i3])

or(logic1,logic2) returns 1 if one of either logic1 or logic2 is not equal to zero or if they are both unequal to zero.
or(i1,i2), or(i1,i3), or(i2,i2)
This operation can be extended to more than two logical variables using the command any(vector1), that returns 1 if any of the elements of vector1 are nonzero, otherwise it returns 0.
any([i1 i2 i3]), any([i2 i2 i2]), any([i1 i2 i2 i2]),

Used less often in scientific computing is the exclusive or construction xor(logic1,logic2) that returns 1 only if one of logic1 or logic2 is nonzero, but not both.
xor(i1,i1), xor(i2,i2), xor(i1,i2)

We use these relational operations to decide whether to perform a block of code using an if structure that has the general form.

```
logictest1 = 0; logictest2 = 1; logictest3 = 0;
if(logictest1)
disp('Executing block 1');
elseif(logictest2)
disp('Executing block 2');
elseif(logictest3)
disp('Executing block 3');
else
disp('Execute end block');
end
```

The last block of code is executed if none of the ones before it has been performed.

```
logictest1 = 0; logictest2 = 0; logictest3 = 0;
if(logictest1)
disp('Executing block 1');
elseif(logictest2)
disp('Executing block 2');
elseif(logictest3)
disp('Executing block 3');
else
disp('Execute end block');
end
```

An if loop will not execute more than one block of code. If more than one logictest variable is not equal to zero, then the first one it encounters is the one it performs.

```
logictest1 = 0; logictest2 = 1; logictest3 = 1;
if(logictest1)
disp('Executing block 1');
elseif(logictest2)
disp('Executing block 2');
elseif(logictest3)
disp('Executing block 3');
else
disp('Execute end block');
end
```

If structures are often used in conjunction with for loops. For example, the following routine adds the components of a vector to the principal diagonal of a matrix that is the sum of two matrices A and B.

```
A = [1 2 3; 4 5 6; 7 8 9];
B = [11 12 13; 14 15 16; 17 18 19];
u = [10 10 10];
C=zeros(3);
for i=1:3
for j=1:3
if(i==j)
C(i,j) = A(i,j) + B(i,j) + u(i);
else
C(i,j) = A(i,j) + B(i,j);
end
end
end
```

As an alternative to if blocks, case structures can be used to chose among various alternatives.

```
for i=1:4
switch i;
case {1}
disp('i is one');
case {2}
disp('i is two');
case {3}
disp('i is three');
otherwise
disp('i is not one, two, or three');
end
end
```

```
clear all
```

2.3. while loops and control statements

A WHILE loops performs a block of code as long as the logical test expression returns a non-zero value.

```
error = 283.4;
tol = 1;
factor = 0.9;
while (error > tol)
error = factor*error;
disp(error)
end
```

If factor >= 1, then the value of error will increase and the while loop will not terminate. A better way, in general, to accomplish the job above is to use a for loop to place an upper limit to the number of iterations that will be performed. A "break" command stops the iteration of the most deeply nested for loop and is called when the condition (error < tol) is reached.

```
error = 283.4;
tol = 1;
factor = 0.9;
iter_max = 10000;
iflag = 0; signifies goal not reached
for iter=1:iter_max
if(error <= tol)
iflag = 1; signifies goal reached
break;
end
```

```
error = factor*error;
disp(error)
end
```
if(iflag==0) write message saying that goal not reached.
```
disp('Goal not reached');
disp(['error = ' num2str(error)]);
disp(['tol = ',num2str(tol)]);
end

clear all
```

2.4. screen input/output

In MATLAB, the basic command to write output to the screen is "disp".
```
disp('The disp command writes a character string to the screen.');
```

When writing integer or real numbers to the screen, the "int2str" and "num2str" commands should be used (for more details see chapter 1 of the tutorial.
```
i = 2934;
x = 83.3847;
disp(['i = ' int2str(i)]);
disp(['x = ' num2str(i)]);
```

The standard command for allowing the user to input data from the keyboard is "input".
```
i = input('Input integer i : ');
x = input('Input real x : ');
v = input('Input vector v : ');
```
try typing [1 2 3]
```
i, x, v

clear all
```