Numerical Methods ITGS219

Lecture: Loops and Conditional Statements:

By: Zahra A. Elashaal

Loops and Conditional Statements:

We now consider how MATLAB can be used to repeat an operation many times and how decisions are taken.

Loops Structures

- The basic MATLAB loop command is **for** and it uses the idea of repeating an operation for all the elements of a vector. A simple example helps to illustrate this:

```
% looping.m
%
N = 5;
for ii = 1:N
    disp([int2str(ii) ' squared equals ' int2str(ii^2)])
end
```

```
This gives the output
1 squared equals 1
2 squared equals 4
3 squared equals 9
4 squared equals 16
5 squared equals 25
```

Loops and Conditional Statements

Example 3.1 The following code writes out the seven times table up to ten times seven**.**

```
str = ' times seven is ';
for j  = 1:10
     x = 7 * j ;
     disp([int2str(j) str int2str(x)])
end
```

- The start of the for loop on the third line tells us the variable j is to run from 1 to 10 (in steps of the default value of 1), and the commands in the for loop are to be repeated for these values.

Example 3.2 The following code prints out the value of the integers from 1 to 20 (inclusive) and their prime factors. To calculate the prime factors of an integer we use the MATLAB command ***factor***

```
for i = 1:20
     disp([i factor(i)])
end
```

```
1    1
2    2
3    3
4    2    2
5    5
6    2    3
7    7
8    2    2    2
9    3    3
10    2    5
11    11
12    2    2    3
13    13
14    2    7
15    3    5
16    2    2    2    2
17    17
18    2    3    3
19    19
20    2    2    5
```

Loops and Conditional Statements

The values for which the for loop is evaluated do not need to be specified inline, instead they could be set before the actual for statement. For example

```
r = 1:3:19;
for ii = r
       disp(ii)
end
```

- displays the elements of the vector r one at a time, that is 1, 4, 7, 10, 13, 16 and 19;

Example 3.3 Suppose we want to calculate the quantity six factorial ($6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1$) using MATLAB.

- One possible way is

```
fact = 1;
for i = 2:6
       fact = fact * i;
end
```

Loops and Conditional Statements

Example 3.4 Calculate the expression ^nC_m for a variety of values of n and m. *This is read as 'n choose m'* and is the number of ways of choosing m objects from n. The mathematical expression for it is

$$^nC_m = \frac{n!}{m!(n-m)!}.$$

- We could rush in and work out the three factorials in the expression, or we could try to be a little more elegant. Let's consider $n!/(n-m)! = n\times(n-1)\times(n-2)\times\cdots\times(n-m+1)$. We can therefore use the loop structure.

```
prod = 1;
mfact = 1;
for i = 0:(m-1)
    mfact = mfact * (i+1);
    prod = prod * (n-i);
end
soln = prod/mfact;
```

Breaking up the calculation like this can lead to problems for large values of m and so it is often best to work out the answer directly:

```
soln = 1;
for i = 0:(m-1)
    soln = soln * (n-i) / (i+1);
end

%This product could also be written as:
soln = 1;
for i = 0:(m-1)
    soln = soln * (n-i) / (m-i);
end
```

Loops and Conditional Statements

Example 3.5 Determine the sum of the geometric progression

$$\sum_{n=1}^{6} 2^n.$$

- This is accomplished using the code:

```
total = 0
for n = 1:6
    total = total + 2ˆn;
end
```

- which gives the answer 126,

Summing Series

In the Example we have summed a series: we now describe this topic in more detail. We start by constructing a code to evaluate.

$$\sum_{i=1}^{N} i^2.$$

```
N = input('Enter the number of terms required: ');
s = 0;
for i = 1:N
    s = s + iˆ2;
end
disp(['Sum of the first ' int2str(N) ... ' squares is ' int2str(s)])
```

Sums of Series

Sums of Series of the Form

$$\sum_{j=1}^{N} j^p,\ p \in \mathbb{N}$$

```
% Summing series
N = input('Please enter the number of terms required ');
p = input('Please enter the power ');
sums = 0;
for j = 1:N
    sums = sums + j^p;
end
disp(['Sum of the first ' int2str(N) ... ' integers raised to the power ' int2str(p) ' is ' int2str(sums)])
```

Sums of Series … Cont.

$$\sum_{j=1}^{N} j^p,\ p \in \mathbb{N}$$

- We note the formula when $p = 1$ is given by $\sum_{j=1}^{N} j^1 = N(N + 1)/2.$
- If we substitute in the values $N = 1$, $N = 2$ and $N = 3$ (all for $p = 1$) we would obtain 3 points on the *'curve'* and these will uniquely determine its coefficients (assuming it is a quadratic).
- We assume that $S_N = aN^2 + bN + c$ and use the three values above to give three simultaneous equations from which we can determine the coefficients a, b and c, these equations are:

$$\begin{array}{lll} N = 1 & a + b + c = S_1 = 1, & (1) \\ N = 2 & 4a + 2b + c = S_2 = 1+2 = 3, & (2) \\ N = 3 & 9a + 3b + c = S_3 = 1 + 2 + 3 = 6. & (3) \end{array}$$

From (1) we see that $c = 1 - a - b$ and this can be substituted into the other two equations to yield:

$$\begin{array}{ll} 3a + b = 2, & (4) \\ 8a + 2b = 5. & (5) \end{array}$$

Now using (4) in (5) we find $a = 1/2$ and then in (4), $b = 1/2$ and finally using (1) we have $c = 0.$ Hence

$$S_N = \frac{1}{2}N^2 + \frac{1}{2}N, \quad \text{for} \quad N = 1, 2, 3.$$

It seems reasonable to expect that the sum for a certain power of p will be of degree $p+1$. In order to determine the *coefficients of a polynomial of degree $p + 1$* we *require $p + 2$ points.*

Sums of Series … Cont.

$$\sum_{j=1}^{N} j^p, \; p \in \mathbb{N}$$

It seems reasonable to expect that the sum for a certain power of *p* will be of degree *p+1*. In order to determine the *coefficients of a polynomial of degree p + 1* we *require p + 2 points.*

```
clear all
format rat
p = input('Please enter the power you require ');
points = p+2;
n = 1:points;
 for N=n
   sums(N)=0;
     for j = 1:N
         sums(N) = sums(N) + j^p;
     end
 end
[coe] = polyfit(n,sums,p+1)
format
```

polyfit(x,y,n) which returns the coefficients of order *n* through the points in (*x, y).*
For the examples *p = 2 and p = 3 we have*

```
>> sumser2
Please enter the power you require 2
coe =  1/6 1/2 1/3
>> sumser2
Please enter the power you require 3
coe = 1/4 1/2 1/4
```

The series for p=2 and p=3 are:

$$\sum_{j=1}^{N} j^2 = \frac{N^3}{3} + \frac{N^2}{2} + \frac{N}{6} = \frac{N}{6}(2N+1)(N+1),$$

$$\sum_{j=1}^{N} j^3 = \frac{N^4}{4} + \frac{N^3}{2} + \frac{N^2}{4} = \frac{N^2}{4}(N+1)^2.$$

Summing Series Using MATLAB Specific Commands:

MATLAB is very good at this type of exercise. Consider the previous example

$$\sum_{i=1}^{10} i^2.$$

- Firstly we set up a vector running from one to ten: i = 1:10;
- and now a vector which contains the values in i squared: i_squared = i.^2;
- Now we use the MATLAB command sum to evaluate this:
- value = sum(i_squared)
- The full code for this example is

```
i = 1:10;
i_squared = i.^2;
value = sum(i_squared)
```

- This can all be contracted on to one line sum((1:10).^2): you should make sure you know how this works!
- Using the command **sum** allows us to simplify our codes: however it is essential we understand exactly what it is doing.

Loops Within Loops (Nested) :

- Many algorithms require us to use nested loops (loops within loops), as in the example of summing series. We illustrate this using a simple example of constructing an array of numbers:

```
for ii = 1:3
    for jj = 1:3
        a(ii,jj) = ii*jj;
    end
end
```

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}$$

- Notice that the inner loop (that is, the one in terms of the variable jj) is executed three times with ii equal to 1, 2 and then 3.

Example 3.9 Calculate the summations $\sum_{j=1}^{N} j^p$

```
N = 6;
for p = 1:3
  sums(p) = 0.0;
   for j = 1:N
     sums(p) = sums(p)+j^p;
  end
end
disp(sums)
```

output

```
21   91   441
```

Conditional Statements:

MATLAB has a very rich vocabulary when it comes to conditional operations but we shall start with the one which is common to many programming languages.

- This is the **if** command which takes the form =>

```
if  (expression)
     commands
       ...
end
```

a < b True if a is less than b
a <= b True if a is less than or equal to b
a > b True if a is greater than or equal to b
a >= b True if a is greater than or equal to b
a == b True if a is equal to b
a ~= b True if a is not equal to b

More often than not we will need to form compound statements, comprising more than one condition.

This is done by using logical expressions, these are:

and(a,b)	**a & b**	Logical AND
or(a,b)	**a | b**	Logical OR
not(a)	**~a**	Logical NOT
xor(a,b)		Logical exclusive OR

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

XOR	0	1
0	0	1
1	1	0

exclusive OR is odd gate

A	B	C	XOR
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

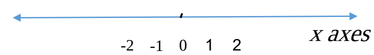
Conditional Statements:

Example 3.10 Determine the sets for which these statements are true; where X represent the x axes.

x>1 & x<2

This is the open set (1, 2).

x<0 | x>=1



the set is $(-\infty, 0) \cup [1,\infty)$.

x>1 | x<2

the answer is $(-\infty,\infty)$.

x<=1 | x>=1

the answer is $(-\infty,\infty)$.

x<=1 & x>=1

The answer is the single value one, written as {1}.

~(x>2)

that is the set $(-\infty, 2]$.

(x>1) & (~(x<2))

the solution is $[2,\infty)$.

abs(x-1) < 2

that is $(-1, 3)$.

Constructing Logical Statements:

We shall now actually construct logical arguments which can be used in if statements.

Example 3.11 Let us consider a command which is only executed if a value x lies between 1 and 2 or it is greater than or equal to 4. We shall try to describe the thought processes involved:

- In order that a value lies between one and two, it has to be greater than one AND less than two, so this component is written as:

  (x>1) & (x<2)

- If x is greater than or equal to 4, which is written simply as x>=4.
- Finally, we need to combine these conditions and this is done using the logical operation OR, since the value of x could lie in one or the other of the regions.
- Hence we have

  ((x>1) & (x<2)) | (x>=4)

- We could use the commands in a different form

```
a = and(x>1,x<2);
b = (x>=4);
c = or(a,b)
```

by making use of the AND and OR commands. Notice here we have actually set "Boolean" variables a, b and c (in fact they are only normal variables which take the values zero or one).

if and elseif Statement

- It is convenient at this stage to introduce some of the other commands which are available to us when constructing conditional statements, namely **if** and **elseif**.
- The general form of these is given by:

```
if (expression)
     commands ...
elseif (expression)
    commands ...
else
     commands ...
end
```

Example 3.12 Consider the following piece of code which determines which numbers between 2 and 9 go into a specified integer exactly:

```
str = 'Divisible by ';
x = input('Number to test: ');
for j = 2:9
      if rem(x,j) == 0
           disp([str int2str(j)])
      end
end
```

if and elseif Statement … Cont.

Example 3.13 Here we construct a conditional statement which evaluates the function:

$$f(x) = \begin{cases} 0 & x < 0 \\ x & 0 \leqslant x \leqslant 1 \\ 2 - x & 1 < x \leqslant 2 \\ 0 & x > 2 \end{cases}$$

One of the possible solutions to this problem is:

```
if x >= 0 & x <= 1
     f = x;
elseif  x > 1 & x <= 2
    f = 2-x;
else
    f = 0;
end
```

Example 3.14 (Nested if statements) The ideas behind nested if statements is made clear by the following example

```
if  raining
   if money_available > 20
            party
   elseif money_available > 10
           cinema
   else
           comedy_night_on_telly
   end
else
   if temperature > 70 & money_available> 40
        beach_bbq
    elseif temperature > 70
        beach
    else
        you_must_be_in_the_UK
   end
end
```

Switch Statement

The MATLAB Command **switch** takes the form:

```
switch switch_expr
    case case_expr1
        commands ...
    case {case_expr2,case_expr3}
        commands ...
    otherwise
        commands ...
end
```

The example given in the manual documentation for this command:

```
switch  lower(METHOD)
        case {'linear','bilinear'}
            disp('Method is linear')
        case 'cubic'
            disp('Method is cubic')
        case 'nearest'
            disp('Method is nearest')
        otherwise
            disp('Unknown method.')
end
```

Example for **switch** command

```
msg = 'Enter first three letters of the month: ';
month = input(msg,'s');
month = month(1:3); % Just use the first three letters
if lower(month)=='feb'
   leap = input('Is it a leap year (y/n): ','s');
end
switch lower(month)
    case {'sep','apr','jun','nov'}
        days = 30;
    case 'feb'
        switch lower(leap)
            case 'y'
                days = 29;
            otherwise
                days = 28;
        end
    otherwise
        days = 31;
end
```

Conditional loops

Suppose we now want to repeat a loop until a certain condition is satisfied. This is achieved by making use of the MATLAB command **while**, which has the syntax

```
while (condition)
        commands...
end
```

Example 3.16 Write out the values of x^2 for all positive integer values x such that x^3 < 2000. To do this we will use the code

```
x = 1;
while x^3 < 2000
    disp(x^2)
    x = x+1;
end
value = floor((2000)^(1/3))^2;
```

Example 3.17 Consider the one-dimensional map:

$$x_{n+1} = \frac{x_n}{2} + \frac{3}{2x_n},$$

subject to the initial condition $x_n = 1$. Let's determine what happens as **n** increases.
We note that the fixed points of this map, that is the points where $\mathbf{x_{n+1} = x_n}$, are given by the solutions of the previous equation:
which are x_n **= ±√3**. We can use the code

```
xold = 2; xnew = 1;
while abs(xnew-xold) > 1e-5
    xold = xnew;
    xnew = xnew/2+3/(2*xnew);
end
```

This checks to see if $x_{n+1} = x_n$ to within a certain tolerance. This procedure gives a reasonable approximation to √3 and would improve if the tolerance (1e-5) was reduced.

Conditional loops with The **break** Command

This allows loops to stop when certain conditions are met. For instance, consider the loop structure:

```
x = 1;
while 1 == 1
     x = x+1;
     if x > 10
          break
      end
end
```

- The loop structure while 1==1 ... end is very dangerous since this can give rise to an infinite loop, which will continue to infinity;
- however the **break** command allows control to jump out of the loop.

Error Checking with **warning** and **error** commands

To this point we have assumed the data made available to a code is suitable;

- MATLAB gives us three very useful commands in this context: **break**, **warning** and **error**. The latter two commands allow us to either warn the user of a problem or actually stop the code because of an irretrievable problem, respectively.
- Both the commands warning and error are used with an argument, which is displayed when the command is encountered.

Example 3.20 Let's now write a code which asks the user for an integer and returns the prime factors of that integer.

```
if code_fails
   error(' Irretrievable error ')
elseif  code_problem
   warning(' Results may be suspect ')
end
```

```
msg = 'Please enter a positive integer: ';
msg0 = 'You entered zero';
msg1 = 'You failed to enter an integer';
msg2 = 'You entered a negative integer';
x = input(msg);
if x==0
      error(msg0)
end
if round(x)~= x
      error(msg1)
end
if sign(x)==-1
      warning(msg2)
x = -x;
end
disp(factor(x))
```

Any Question?