الجامعة المستنصرية

كلية الهندسة

قسم هندسة الموارد المائية

Lecture 6: Logical Functions

المحاضرة السابعة: الدوال المنطقية

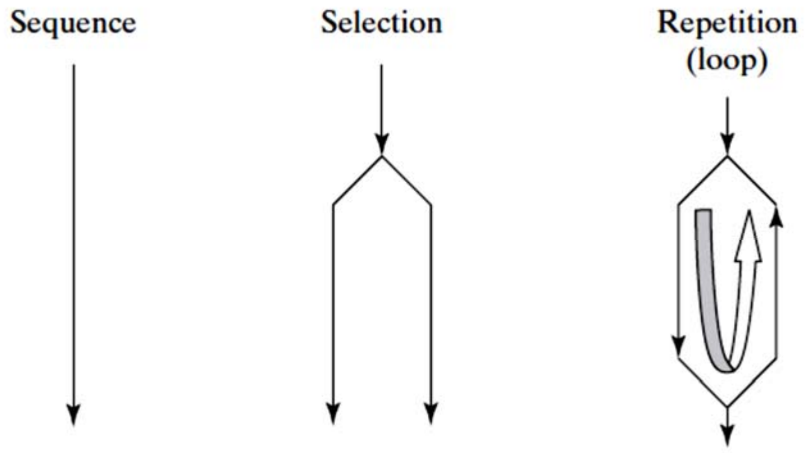
المنهاج الدراسي لمادة البرمجه والتطبيقات (ماتلاب)

الكورس الدراسي الاول / المرحلة الثانية

من اعداد د. صباح حسن لفته

Introduction

In MATLAB, sections of script code can be written as repetition statements, sequences, and selection structures. The following figure shows the three means structures that used widely in MATLAB codes.



In previous lecture we discussed the repetition statements, now we will demonstrate the others which are the ***sequences*** and ***selection structures***. A **sequence** is a list of commands that are executed one after another. While a **selection structure** allows the programmer to execute one command (or set of commands) if some criterion is true and a second command (or set of commands) if the criterion is false. A selection statement provides the means of choosing between these paths, based on a ***logical condition***. The conditions that are evaluated often contain both ***relational*** and ***logical*** operators or functions.

a- Relational and Logical Operators

The selection and repetition structures used in MATLAB depend on relational and logical operators. MATLAB has six relational operators for comparing two matrices of equal size, as shown in the below table.

Relational Operator	**Description**
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
~=	not equal to

For example, let say we have two scalars,

```
x=3;
y=5;
```

in this case, x has value smaller than y and if we use the relational operator (>)which means greater than to compare between x and y in MATLAB,

```
x > y
```

the result of comparison I either true or false. In this case, x is not greater than y and MATLAB will return,

```
ans=  0
```

indicating that the comparison is false. MATLAB uses this answer in selection statements and in repetition structures to make decisions. For matrices, the same procedure is followed, so let say the x and y are matrices,

```
x=1:5;
y=x-3;
x=[ 1 2 3 4 5 ]
y=[-2 -1 0 1 2]
```

and the comparison,

```
x > y
```

MATLAB answer will be,

```
ans= 0  0   0   0   0
```

Hint: for a comparison to be true for an entire matrix, it must be true for *every* element in the matrix. In other words, all the results must be ones.

On the other hand, MATLAB also allows to combine comparisons with the logical operators such as ***and***, ***not***, and ***or*** as shown in below table,

Logical Operator	Description
&	and
~	not
|	or

For example, the following code,

```
x = [ 1, 2, 3, 4, 5];
y = [-2, 0, 2, 4, 6];
z = [ 8, 8, 8, 8, 8];
z>x & z>y
```

MATLAB will return the results,

```
ans =
     1  1  1  1  1
```

because **z** is greater than both **x** and **y** for every element. The statement

```
x>y | x>z
```

is read as "**x** is greater than **y** or **x** is greater than **z**" and returns,

```
ans =
     1  1  1  0  0
```

This result means that the condition is true for the first three elements and false for the last two.

b- Logical Functions

MATLAB offers both traditional selection structures, such as the family of **if** functions, and a series of logical functions that perform much the same task. The primary logical function is ***find***, which can often be used in place of both traditional selection structures and loops.

The ***find*** command searches a matrix and identifies which elements in that matrix meet a given criterion. For example, the Iraqi Army Academy requires applicants to be at least 170 cm tall. Consider this list of applicant heights:

```
height = [160, 170, 165, 183, 175, 198, 191]
```

You can find the index numbers of the elements that meet our criterion by using the find command:

```
accept = find(height>=170)
```

This command returns,

```
accept =

          2    4    5   6    7
```

The find function returns the index numbers from the matrix that meet the criterion. If you want to know what the actual heights are, you can call each element, using the index number:

```
height(accept)
ans =
          170  183  175  198  191
```

An alternative approach would be to place the commands in one statement,

```
height(find(height(>=170)))
```

We could also determine which applicants do *not* meet the criterion. Use

```
  decline = find(height<170)
```

which gives

```
   decline =

                 1     3
```

To create a more readable report, use the ***disp*** and ***fprintf*** functions:

```
disp('The following candidates meet the height requirement');
fprintf('Candidate # %4.0f is %4.0f cm tall \n', [accept;height(accept)])
```

These commands return the following table in the command window:

```
The following candidates meet the height requirement

     Candidate #    2 is  170 cm tall

     Candidate #    4 is  183 cm tall

     Candidate #    5 is  175 cm tall

     Candidate #    6 is  198 cm tall

     Candidate #    7 is  191 cm tall
```

Likely, we could also create a table of those who do not meet the requirement:

```
disp('The following candidates do not meet the height requirement')
fprintf('Candidate # %4.0f is %4.0f cm tall \n', [decline;height(decline)])
```

Similar to the previous code, the following table is returned in the command window:

```
The following candidates do not meet the height requirement
     Candidate #     1 is 160 cm tall

     Candidate #     3 is 165 cm tall
```

Most of the time, the find command can and should be used instead of an ***if statement***. In some situations, however, the ***if statement*** is required. There are three types of ***if statement***:

Simple if which has the following form,

```
>> if  comparison
     statements
>> end
```

If the comparison (a logical expression) is true, the statements between the ***if statement*** and the ***end statement*** are executed. If the comparison is false, the program jumps immediately to the statement following end.

The if/Else Structure we saw that simple if statement allows us to execute a series of statements if a condition is true and to skip those steps if the condition is false. Adding the else clause allows us to execute one set of statements if the comparison is true and a different set if the comparison is false. The if /else statement has the following form,

```
>> if  comparison
     statements
>> else
     statements
>> end
```

The Elseif Structure in some cases we need to arrange several levels of if/else statements, in this case it may be difficult to determine which logical expressions must be true (or false) in order to execute each set of statements. The elseif function allows us to check multiple criteria while keeping the code easy to read. The elseif statement has the following form,

```
>> if  comparison
     statements
>> elseif comparison
     statements
>> else
     statements
>> end
```

Let explain this in example, consider the following lines of code that evaluate whether to issue a driver's license, based on the applicant's age:

```
if age<16
disp('Sorry – You'll have to wait')
elseif age<18
disp('You may have a youth license')
elseif age<70
disp('You may have a standard license')
else
disp('Drivers over 70 require a special license')
end
```

In this example, MATLAB first checks to see if **age** < **16.** If the comparison is true, the program executes the next line or set of lines, displays the message **Sorry_You'll have to wait**, and then exits the if structure. If the comparison is false, MATLAB moves on to the next elseif comparison, checking to see if age < 18 this time. The program continues through the if structure until it finally finds a true comparison or until it encounters the else. Notice that the else line does not include a comparison, since it executes if the elseif immediately before it is false.

Example (1): Write a script code to find the value of (x) of the following problem.

$$x = \begin{cases} 1 \qquad ab \quad for\ a > 5 \\ \dfrac{3}{2} \qquad ab \quad for\ a \le 5 \end{cases}$$

where b=15

```
>>%% Writing a code to find the value of x
>> %% Input known values
>> b=15;
>>%% Using for loop and if/else statement
>>for a=1:10
>> if a>5
>> x(a)=a*b
>> else
>> x(a)=(2/3)*(a*b);
>> end
>>end
```

MATLAB will return the following results,

```
x =
7.5   15   22.5   30   37.5   90  105  120  135  150
```

Example (2): Write a script code to solve the following equation.

$$x = \begin{cases} \frac{\sin(x)}{x} & x \neq 0 \\ 1 & x = 0 \end{cases}$$

```
>>%% Writing a code to solve the equation
>> x=-5;
>> for i=1:11
>> if x==0
>>y(i)=1;
>>else
>>y(i)=sin(x)/x;
>>end
>> x=x+1;
>>end
```

The result is,

```
y =
  -0.1918  -0.1892   0.0470   0.4546   0.8415   1.0000   0.8415   0.4546   0.0470
-0.1892
```

Example (3): Write a MATALB code to find the variable Z of the following equations.

$$\boldsymbol{Z} = \begin{cases} \mathbf{5} & \boldsymbol{if\ v < 0} \\ \mathbf{10} + \boldsymbol{v} & \boldsymbol{if\ v = 0} \\ \mathbf{6} - \boldsymbol{v} & \boldsymbol{if\ \ v > 0} \end{cases}$$

where: v=-20:5:20

```
>>%%% The code written to find the value of Z
>> v=-20;
>> for i=1:9
>> if v<0
>> Z(i)=5;
>> elseif v==0
>> Z(i)=10+v;
>> else
>> Z(i)=6-v;
>> end
>> v=v+5;
>>end
```

The result will be,

```
Z =
   5    5    5    5   10    1   -4   -9  -14
```

Example (4): Write a MATALB code to find maximum an minimum values of matrix (a) where a=[11 3 14; 8 6 2; 10 13 1], don’t use max. and min. commands.

```
>>%% Writing MATLAB code to find min and max values
>> a=[11 3 14; 8 6 2; 10 13 1];
>>large=a(1,1);
>> small=a(1,1);
>>[n,m]=size(a);
>>for i=1:n
>>for j=i:m
>> if a(i,j)> large
>> large=a(i,j);
>>elseif a(i,j)<small
>> small=a(i,j);
>> end
```

```
>>end
>>end
```

The result,

large =   14

small =   1