Introduction to MATLAB-3

For B.Sc. Sem-4 ELTA Sec-2

Script files

- In this section we show you how to store commands in a file. Such MATLAB command files are known as *script files*. They are stored with the .m extension. In MATLAB files with the .m extension are also called M-files. All script files are M-files, but not all M-files are script files as you will see later (for example, MATLAB functions are also stored in .m files)
- From the menu choose *File* then *New* then *M-file*.
- You are now in the *MATLAB Editor/Debugger*.
- Type the following lines in the new file window. Note that the file is still called 'Untitled'.
- **% Add two matrices and display the result A = [1, 1; 2, 2]**
- **B = [2, 2; 3, 3]**
- **D = A + B**
- **disp('The result is '); disp(D);**

Script files

- The first line is a comment written to explain what your program does. MATLAB ignores everything on a line after a %-character. Note that comments are displayed in a different colour so you can directly recognize them. It is very important to write adequate comments so that other people who need to use your program can understand it.
-
- You may have noticed that when closing brackets the two matching brackets are identified (usually by flashing or underscoring them). This is useful in cases where you have several sets of brackets: you can easily check if all brackets are matched up.
-
- The **disp** command is used to display output strings on your screen, as in **disp('the result is ')** and also to display variables as in **disp(D)** . The text that will be displayed is shown in a different colour. Note that we can display strings or variables but not at the same time. So the command **disp('the result is ',D);** is not accepted.

Save this file.

- We will now save this file.
- Choose *File* then *Save As* from the menu. The current directory (which should be the directory you just created) is automatically selected. Save the file as addmat.m .
- Go back to the command window and type **addmat** to run the file (note that you should not type the
- .m extension!). If MATLAB gives an error saying that it can not find addmat, make sure that the directory in which the addmat.m file is stored and MATLAB's current directory are one and the same.

Relational operators

- MATLAB has 6 relational operators to make comparisons between variables. These are
-
- **<** *is less than*
- **<=** *is less than or equal to*
- **>** *is greater than*
- **>=** *is greater than or equal to*
- **==** *is equal to*
- **~=** *is not equal to*
-
- Note that the *is equal to* operator consists of two equal signs and not a single = sign as you might expect. We'll play around with these relational operators for a little while.

Relational operators

- Go back to the Command Window. Set up the scalars q=10, w=10, and e=20.
- In MATLAB a logical expression has two possible values that are not 'true' or 'false' but *numeric*, i.e. 1 if the expression is true and 0 if it is false.
- Type **w< e** to see that the result is given the value of 1 because w is indeed less than e. Now type **q==e**. The == operator checks if two variables have the same value. Therefore the answer is 0 in this case.
- Relational operations are performed *after* arithmetic operations. For example **q == w-e** results in 0. Parentheses can be used to override the natural order of precedence. So **(q==w) - e** results in -19.
- The relational operators can be used for all elements of vectors simultaneously.
- Create the vectors x and y by typing **x=[-1, 3, 9]** and **y=[-5, 5, 9]** . Now type
- **z= (x < y)** . The i-th element of z is 1 if x(i) < y(i), otherwise it is 0. So, the answer is the vector with elements 0, 1, and 0.

Logical operators

- MATLAB uses three logical operators. These are
- and **&**
- or **|**
- not **~**
  For **&** (and) to give a true result both expressions either side of the **&** must be true. The logical expression
- **e> 0** is true and the logical expression **q<0** is false. So the logical expression (**e>0) & (q<0**) is false. For **|** (or) to give a true result only one of the expressions either side of the **|** needs to be true.
- Type (**e>0) | (q<0)** . Is the result as expected?
-

Logical operators

- The **~** (not) operator changes a logical expression from 0 to 1 and vice versa. So **result = ~(q<0)** would be 1. The operator ~ has a high priority. So, to avoid mistakes, put whatever expression you want to negate in parentheses, as we did here.
- The operations **&** and **|** are performed after the relational operations,**<>==** etc.
- Just as with relational operators we can use logical operators for vectors or matrices.

The ‘if' statements

- The if-statement starts with **if** followed by a condition, usually a relational operation, and is finished with an
- **end;** command. For example (taken from a soccer game simulator)
- **if score ~= 0**
- **disp('GGOOAAALLL – the score is '); disp(score);**
- **end;**
- The body of this if-statement consists of two lines and both will be carried out if the score is not equal to zero. We can include as many lines in a body as we want.

‘If’ with else statement

- We could also write a message if the score is zero. Instead of writing a separate if-statement for this case we combine it with the previous one in
- **if score ~= 0**
- **disp('GGOOAAALLL – the score is'); disp(score);**
  else end;

- **disp('darn – kick too soft');**

  If the score is not zero the first body will be carried out (the one immediately following the first condition statement). Otherwise the command in the body after the else-statement is carried out.

Using ‘elseif’ statement

- **if score == 1**
- **disp('FFFIIIRRRSSSTTT GGGOOOAAALLL');**
- **elseif score > 1**
- **disp('We are on a roll');**
  else end;

- **disp('darn – kick too soft');**
  The **elseif** statement allows us to include more options.

Nested ‘if’ Statement

- another example where now we check if a penalty kick will enter the goal (our keeper has a reaction time of 0.5 seconds):
- **if time < 0.5**
- **% compute the height of the ball as it reaches the goal**
  height = initialS*time*sin(angle) – 0.5*g*time^2;
- **score = score + 1;**
- **end;**
- **end;**
- From this example we see that it is possible to have nested if-statements, that is if- statements inside other if-statements. Now we get two **end** statements, one for each **if**. It is VERY IMPORTANT TO USE INDENTATION!!! Otherwise it is hard to figure out which **end** belongs to which **if**.

The 'for' loop

- Let’s start with a very simple example. We will create a vector with 3 random angles between 0 and !, compute the cosine of each angle, and display it.
- The program is
- **angle = pi*rand(3,1);**
- **for k=1:3**

  cosangle = cos(angle(k));

  disp(cosangle);
- **end;**
- Note
- k is called the loop-variable.
- the loop is ended as required by the **end;** statement
- the body is the code between the **for**-statement and the corresponding **end;**
- the body is indented to make it easy to see what’s happening inside the loop

Using ‘for’ loop

- Suppose we want to find the maximum element of a vector x with 4 elements that are random numbers between 0 and 1.
- We store the maximum value in the variable called maxval. At the start of our code we initialize maxval to
- Then we access all elements of the vector x one-by-one and each time check if the element is larger than maxval. If so, we set maxval to this new value. Accessing the elements can be done in a for-loop:
- **x = rand(4,1); maxval = 0;**
- **for k=1:4**
- **if (x(k) > maxval)**
- **maxval = x(k);**
- **end; end;**
- **disp(maxval);**
- Note that both the if-statement and the for-loop must have a corresponding end statement. We used indentation and therefore it is easy to see which end belongs to which statement.

Nested for-loops

- Just as we can have if-statements inside for-loops we can have for-loops inside for-loops. We compute the exponential of each element of a 2x2 matrix A.

```
A = rand(2,2);
for k=1:2
    for m=1:2
        exponential = exp(A(k,m));
    display(exponential);
```

- ```
    end;
```
- ```
end;
```

Explanation of nested for loop

1. What happens? Let's go through the program step-by-step. You can do this in the debugger as well. We will call the k-loop the outer loop and the m-loop the inner loop.
2. MATLAB enters the outer loop and sets k to 1.
3. At the next line, MATLAB reaches the inner for-loop and sets m to 1.
4. It computes the exponential of A(1,1) and displays it.
5. It reaches the **end;** statement corresponding to the inner-loop.
6. Since it has not completed the inner loop yet, MATLAB jumps back to the **for**-statement of the inner loop and sets m to 2.
7. It computes the exponential of A(1,2) and displays it.

Explanation of nested for loop

8. It reaches the **end;** statement of the inner-loop.
9. It realizes that it has completely carried out the inner loop and continues to the next line in the program which is the **end;** statement of the outer-loop.
10. Since it has not completed the outer loop yet, MATLAB jumps back to the **for**-statement of the outer loop and sets k to 2.
11. It goes to the inner loop and passes through it twice as above for k=1, computing the exponential of A(2,1) and displaying it, and then computing A(2,2) and displaying it.
12. It then reaches the **end;** statement of the outer loop once more, realizes it is done and stops.

‘While’ loops

- In a for-loop we repeat the statements inside the body a fixed number of times. But often we want to repeat some calculation not for a fixed number of times but until some condition is satisfied. Let's look at a very simple example.

```
x = 7;
while x > 0
        x = x-4;
        disp(x);
end;
disp('Done');
```

How does ‘while’ loop works?

1. The variable x is equal to 7 initially.
2. Since x > 0 MATLAB enters the body of the while loop and subtracts 4 from x, which is now equal to
3. MATLAB displays the new value of x.
4. MATLAB reaches the **end;** statement and jumps back to the while-condition (**while x>0**).
5. Again x > 0 so MATLAB goes into the while-loop, sets x to –1 and displays it.
6. MATLAB reaches the **end;** statement and jumps back to the while-condition.
7. Now x is smaller than 0, so MATLAB will not enter the body of the while-loop but jumps to the final display statement and writes "Done" to screen.

In this example we must set x to a value (initialize) before we start the while-loop. Otherwise the condition in the while-statement can not be checked. So, MAKE SURE YOU INITIALIZE.