MATLAB for MAPH 3071 Lab 2

1. Iteration Method - Continued

We have looked at **for** loops as a control structure. We will now introduce more ways to control iterations.

While Loops:

MATLAB provides another type of *loop,* the **while** command, which is very useful for mathematical computation. This simple example will give you an idea of how it works.

```
» x = 10;
» while (x > 1)
x = x/2
end
```

The first line sets x=10. Just like the **for** command, the **while** command is always paired with an end command. The above example has a condition, x > 1, and a command, x = x/2 (i.e. *the new value of x = the old value of x divided by two*). The **while** loop will execute the command as long as the condition is true. In other words this example will start with the number 10, keep on dividing it by two until a number less than one is obtained and then stop.

This idea is very useful for iteration problems. Up to now we have been performing a fixed number of iterations with the **for** command. However, say we wanted to stop the iteration, in the problem x = exp(-x), when we have achieved a certain level of accuracy between successive guesses; i.e. stop when the relative difference between successive approximations is smaller than some chosen tolerance

i.e $|x(i)-x(i-1)|/|x(i)| <$ tolerance (taken from lectures).

We don't know at what stage this will happen so the **for** command is not the best loop to use. However, the **while** command is tailor made for using stopping criteria. The example below stops the iterative solution of x = exp(-x) when the relative difference between successive approximations is less than 1/1000 (the comments after the % symbols are explanations of the various lines).

```
»xold=1.;                          % Our initial guess at the solution.
»tol=.001;                         % Set the tolerance we want for the relative difference.
»rel=1.;                           % Initial value for the rel. diff. (needed for the while command).
» itr=0;                           % will count the number of iterations performed
»while (rel>tol)                   % As long as the rel. diff. is too large, continue with the iteration.
itr=itr+1                          % Updatng the number of iterations performed
xnew=exp(-xold)                    % new approximation = exp(-old approximation).
rel=abs(xnew-xold)/abs(xnew);      % Calculate the relative difference between xnew and xold.
xold=xnew;                         % Reset xold for the next step.
end
```

We have now solved the iteration problem without using an index i, in other words at the end of the calculation we don't have an array x(i) but rather the last two approximations xnew and xold.

As an exercise solve also the nonlinear equation x^2+2*x*sin(x)-1=0 by iteration using the while command - set your tolerance to 0.001.

We have now covered the **for** and **while** commands which are both examples of *loops.* There is one final command which will be very useful; the **if** command. This is an example of a *conditional;* i.e. if we want to do something only when a specified condition is fulfilled. The following examples may seem a little strange at the moment since we haven't (yet !) covered any applications which really need the **if** command. Consider a variable x in a MATLAB routine which has an arbitrary value. Define a second variable y which is equal to 1 if x is greater than zero and -1 if x is less than zero. We could set this up using the if command as follows.

```
» x=5;
»if (x>0)                % If x is greater than zero
y=1                      % then y is equal to 1,
elseif(x<0)              % or else, if x is less than zero,
y=-1                     % then y is equal to -1.
end
```

Repeat this, but this time let x = -5;

As with **for** and **while** the if command is always paired with an end. However, this is not a loop since the commands are executed just once. The second condition is specified with an ***elseif*** (you could also have an example with more than one **elseif**). A slight variation on the above example would be the following,

```
»if (x>0)                % If x is greater than zero
y=1                      % then y is equal to 1,
else                     % otherwise
y=-1                     % y is equal to -1.
end
```

In this instance y will be equal to -1 if x is less than or equal to zero. The usefulness of the if command will become apparent later in the course when we need to use it.

The three commands **for, while** and **if** are the most important control structures we will use in MATLAB. All of our computational problems can be solved using them properly so spending the time trying to understand them is well worth the effort. Also every programming language on a computer (e.g. Fortran, C, Pascal etc.) has commands like **for, while** and **if** although they may have slightly different names. So if you understand the ideas behind these three commands, and become accustomed to using them with MATLAB, then you have grasped some of the essential structures which appear in all programming languages.

Lets use these control structures to improve an algorithm we used earlier. Back to the original problem of finding the solution to $x = e^{-x}$. What we now want to do is to write

an algorithm that will terminate after a specified number of iterations. This is a useful thing to do, since if a algorithm does not reach convergence it may go on for ever !

```
»xold=1;
»tol=.001;
»rel=1;
» itr=0;
» while (rel>tol)
itr=itr+1;
xnew=exp(-xold);
rel=abs(xnew-xold)/abs(xnew);
xold=xnew;
if itr ==5                      % stops after  a maximum of 5 iterations
disp('max iterations reached')  % displays this message if 5 iterations are reached
return                          % stops execution of while loop
end                             % closes if command
end                             % closes while loop
```

Note that this time convergence is not reached as we have set the maximum number of iterations too low (have a look at the value for xnew - this is the last value reached before we exited after 5 iterations. Compare it to the correct answer we got earlier). Also note that we have used the semi-colons here stop MATLAB from printing the result at each iteration - we are only interested in the final answer and whether or not we have reached convergence.

This is an example of a control structure (**if**) nested within another control structure (**while**). This is a powerful way of programming - but you must be careful where you position control structure within loops.

I'm sure at this stage you are tired of typing similar commands into MATLAB. So far we have used MATLAB interactively, typing in commands and getting a result. Now we will learn how to reuse commands, perhaps staring the same algorithm with different initial values.

2. Re-useable programming in MATLAB: M-Files

IMPORTANT: create a folder on your Home Directory called 'mfiles'; ie you should now have a folder h:\mfiles

The last set of commands you typed in worked properly. However, once typed in on the screen it can't be used again so that if we wanted to increase the maximum number of iterations (and we do obviously !), we would have to type in the commands once more. MATLAB provides a facility for creating files (called M-files) where we can store a routine like the one above and then run it whenever we like. **To create an M-file** which would solve the above problem go through the following steps. Using the mouse to select the FILE menu item, then click on NEW (since we want to create a new file) and finally click on M-file. A window will now open on the screen (the *MATLAB editor/debugger*) which is an editor and where we can type in the commands which we want in our file. Type the program below into this editor. This is a slightly modified version of the program we ran interactively earlier.

```
% Iterate - "Numerical Recipe" to solve x=exp(-x) by iteration where
% the initial guess, xold and the maximum number of iterations are
% the required inputs.

clear; help iterate;
xold=input('Enter the initial guess x(1) -');
maxit=input('Enter Maximum No. of Iterations -');
tol=.001;
rel=1;
itr=0;
while (rel>tol)
  itr=itr+1;
  xnew=exp(-xold);
  rel=abs(xnew-xold)/abs(xnew);
  xold=xnew;
  if rel>tol
        result='Algorithm did not converge before exit';
    else
        result='Convergence criteria satisfied';
    end
    if itr ==maxit          % stops after the maximum number of iterations
        disp('max iterations reached')
        disp(result)
    return
  end
end
disp(result)
disp('answer =')
disp(xnew)
return
```

Once these lines are typed in, choose the FILE menu item, then SAVE AS. You will then be asked for the name of the program and to specify a location to save the program to. Type the following in the filename box: **h:\mfiles\iterate.m** (all names must end with .m hence the name M-files) and then press *return* or click on OK. It is a very good idea to give your M-files a relevant name. You now have a file saved in your h:\mfiles folder that you can use repeatedly.

First let us go through the lines in iterate.m in order to understand what it does. The first three lines provide the name of the m-file and a description which can be anything you like (any line beginning with % is recognised by MATLAB as a comment and not a command). These lines are essential so that at some point in the future you can tell immediately what your programme does. The next line has two commands; *clear* which we have come across before and *help iterate* which types the description of the programme (i.e. the first three lines) on the screen when we run it. The fifth line is our initial guess where we have written it so that MATLAB will ask us to choose a number each time we run the programme; this is what the **input** command does. The next line similarly asks us to specify the maximum number of iterations required. The remaining lines are essentially the same as we have described before (there are some minor additions - you should be able to tell what they are doing). Finally **return** must be included so that MATLAB will know when the programme is finished.

To run an M-file: Return to the MATLAB command window (you can use the Alt+Tab key sequence). You now have a program you wish to run, but first you need to tell MATLAB where to look for your program. Type the following to get MATLAB to look in the right folder in your Home Directory for your program;
» addpath h:\mfiles

Now type;

» iterate

to run your program.

To change an M-file (i.e. if you have made a mistake or want to change something) go back to the editor and make your changes. When you're finished select FILE and then SAVE so that iterate.m will be updated. Rerun *iterate.m* a number of times changing the staring value and maximum iterations.

Note on good programming practice;

- Call your programs something meaningful - later on you will have many programs so this is important
- Always write a few lines that describe your programs and have this description displayed when the program runs (as shown in the example above)
- When using loops and other control structures indent the commands within the loops for ease of reading
- Use the % to write notes beside lines of code describing what they do. MATLAB treats anything written on a line following a % as a comment only - i.e. it does not try to interpret it as a command

As an exercise, write an M-file which solves question (8) on the first problem sheet, i.e. find a root of x^3-2*x-5=0 using the Newton method, and which allows you to choose different initial guesses each time. You can give your M-file any name you like ending with .m except iterate !

END