Maths 6141 - Numerical Methods

Computer Lab 3

Computer Lab 3

When using python to implement and test numerical methods, you will construct scripts (just lists of commands, at their most basic) and functions. These functions could be in the scripts or in their own separate file (in which case, to use them you need to `import` the file name). Such a function might start

```
def function1(input1, input2, input3):
```

and end

```
    return output1, output2
```

When implementing a method everybody makes mistakes. The key is to be able to use all available tools to find the mistakes quickly and easily. You will find some suggestions in the Programming Guide on Blackboard, although these are a little Matlab-centric. For documentation, I recommend using the numpydoc standard and looking at their example.

Tasks

On Blackboard you will find a python script called `TestInverse.py`. This is intended to take a matrix A, check that its condition number is less than a tolerance `tol` and determinant bigger than the inverse of `tol`. If both those conditions are met then it should return the inverse of A, otherwise it should give a meaningful error message.
Open the file in the Editor. It does not work; you should make it work properly and document it. Key points include:

1. There is no useful help comment at the top of the function. Try typing (in the Console) `TestInverse?` and see what the result is. Improve the comment so that the function could be used without needing to refer to this sheet.

2. Note that the editor warns where it believes errors exist. Look for all the warning signs given by the editor, and correct the script until it believes there are no remaining syntax errors. Note that there may be errors in the script which are not mere syntax errors.

3. Test if the algorithm works by calling the function, using the trivial data $A = I_2$ and $\text{tol} = 10$, by:

   ```
   > A = np.eye(2,2)
   > tol = 10
   > invA = TestInverse.TestInverse(A, tol)
   ```

   If the algorithm still does not work, **carefully read the error message**. It should, at the very least, tell you the line number where things have gone wrong, isolating the line to fix.

4. Debugging is not complete until you are sure that the function behaves as you expect for *all* input. That means it should fail correctly when the input makes no sense. Try a range of input to check that it works sensibly. It should fail on cases such as

   (a) $A = I_2$, $\text{tol} = 0$,

   (b) A is a vector, such as $(1, 1)$,

(c) A is the zero matrix,

(d) A contains complex numbers or characters,

(e) tol is a vector.

Where the error message is unclear, or it works and gives "incorrect" output, suitable assertion statements should be used.

Next download the file `TestSequence.py`. This is meant to show that

$$\begin{aligned} \lim_{n\to\infty} 2^{-n} &= 0, \\ \sum_{n=0}^{\infty} 2^{-n} &= 2. \end{aligned}$$

It does this by computing the results for n up to N, where N is the input. If you call the function with $N = 50, 100, 200$ you will see that although the limit appears to be correct the sum is not (call the function using e.g. `limit, value = TestSequence.TestSequence(50)`).
Even if the error in the function is obvious to you, it is still worth testing the script using *breakpoints*.

1. Open the script in the Editor. This should show no obvious errors.

2. In the left edge of the window each line is numbered. Wherever there is a command there is a small line next to the number. Click on line 33 - a small red circle should appear, indicating a breakpoint here.

3. Run the script from the `Debug` window. The final line actually calls the function. This should stop execution at the breakpoint.

4. Use the Variable Inspector to look at the values of all of the variables at this point. Consider what you believe they should be.

5. After checking that all values are as you expect you can step forward one command at a time or let the program continue to run until the next breakpoint – check the icons on the Editor toolbar to see what they do.

6. Either by stepping forward multiple times or by putting a breakpoint around line 33, check the values of `limit` and `sumseq` for the first few steps inside the loop. This should show where the error is.