Worksheet 2

All exercises are on material covered in Lecture 2.

There is a quiz for lecture 2 on Keats you can to test yourself on the basic ideas.

I've written this material up in greater detail as the second part of Chapter 2 and the first part of Chapter 3

An alternative reference is to use MATLAB's own tutorial (`https://uk.mathworks.com/support/learn-with-matlab-tutorials.html`). Note that this covers the whole of the MATLAB language, but we only use a small part of the language in this course.

Questions from Chapter 2

1) Use the function `assertApproxEqual` to simplify the function testCumulativeNormal. You should be able to make three different simplifications. Notice how much more readable the code becomes.

(Solution: see the file `testCumulativeNormal.m` in `chapter2.zip`)

2) [⋆] Write a test function for your Black Scholes formula. It should test the "static bound" that the price of a call option is always greater than $S - exp(-rT)K$. It should also check that very near to maturity the price is well approximated by the immediate exercise value. Can you think of any other tests?

(Solution: see the file `testBlackScholesCallPrice.m` in `chapter2.zip`)

3) Use the function `integrateNumerically` to compute $\int_0^1 \sin(s)\,\mathrm{d}s$ and also to compute $\int_1^3 (x^2 - 2x + 2)\,\mathrm{d}x$.

4) Write some automated tests for the function `integrateNumerically`.

(Solution: see the file `testIntegrateNumerically.m` in `chapter2.zip`)

5) [⋆] Write a function `integrateFromMinusInfinity( f, x, N)` which makes the substitution $t = x + 1 - \frac{1}{s}$ and uses this to evaluate the integral $\int_{-\infty}^{x} f(t)\,\mathrm{d}t$

using the rectangle method with N steps. This function should itself call `integrateNumerically`. Test your function. Modify the cumulativeNormal function so that it calls this function.

(Solution: see the file `testIntegrateFromMinusInfinity.m` in `chapter2.zip`)

6) Write a function `normalDensity` which computes the probability density function of the normal distribution. Modify the cumulativeNormal function so that it calls this function.

(Solution: see the file `cumulativeNormalVersion3.m` in `chapter2.zip`)

Questions from Chapter 3

7) Write a function `myProd` to compute the product of all the elements in a vector.

(Solution: see the file `myProd.m` in `chapter3.zip`)

(Solution: see the file `testMyProd.m` in `chapter3.zip`)

8) [⋆⋆] Write a function to find the maximum value in a vector. You are not allowed to use the MATLAB `max`, `min` or `sort` functions!

If you are new to programming, you may find this question difficult. If you struggle, imagine you were given one thousand cards each with a different number printed on it. How would you find the maximum? Write down detailed instructions for how you would do this in English and then try to convert them into MATLAB code.

(Solution: see the file `findMax.m` in `chapter3.zip`)

(Solution: see the file `testFindMax.m` in `chapter3.zip`)

9) [⋆⋆] Modify the `integrateNumerically` function from the last chapter so that it uses a `for` loop rather than a `sum` statement. The benefit of this is that `integrateNumerically` will now work for functions like `cumulativeNormal` that can only process a single argument rather than a vector of values.

(Solution: see the file `integrateNumericallyForLoop.m` in `chapter3.zip`)

10) [⋆] In the game paper-scissors-stone, let the number 0 represent paper, the number 1 represent scissors and the number 2 represent stone.

Write a function `hasPlayerAWon( A, B)` that uses `if` statements to decide who has won given the numbers representing the selections of player A and player B.

(Solution: see the file `hasPlayerAWon.m` in `chapter3.zip`)

11) You can use the value `inf` to represent infinity and the value `-inf` to represent negative infinity in MATLAB.

Given this, write a function `integrateNumericallyVersion2(f, a, b, N)` that allows you to specify infinite values for the integration range `[a,b]`. You will need to perform appropriate substitutions before calling the old function `integrateNumerically` with a finite range.

(Solution: see the file `integrateNumericallyVersion2.m` in `chapter3.zip`)

(Solution: see the file `testIntegrateNumericallyVersion2.m` in `chapter3.zip`)

12) The Fibonnacci sequence is defined by $x_1 = 1$, $x_2 = 1$ and thereafter by $x_n = x_{n-1} + x_{n-2}$. Write a function `fibonnacci(n)` that computes the n-th Fibonnaci number x_n.

13) Write your own function `myisprime` that tests if a number is a prime or not. You can use the function `rem` which computes the remainder of two numbers after a division.

(Solution: see the file `myIsPrime.m` in `chapter3.zip`)

14) [$\star$] Modify the function `blackScholesCallPrice` from the last chapter so that it can take a vector for each parameter and so compute call option prices for a variety of scenarios all with one function call.

(Solution: see the file `blackScholesCallPrice.m` in `chapter3.zip`)

15) Write a function `blackScholesPrice` which behaves like `blackScholesCallPrice` except that it also takes an array of logical values indicating whether the option is a put or a call and prices the option accordingly.

Can you write this code so that it operates on vectors of parameters without using any `for` loops? To do so you will need to vectorize any `if` statements.

(Solution: see the file `blackScholesPrice.m` in `chapter3.zip`)

16) Without using a `for` loop, find the sum of all the numbers $\sin(n)$ where n is between 1 and 100 (inclusive) and $\sin(n)$ is greater than one half.

(Solution: see the file `answerFinalExercise.m` in `chapter3.zip`)