Computational Physics: PHAS0030, 2018-2019
Problem Sheet 1

Hand in your answers by **Monday 28 January**, *at the lectures or on Moodle through the link provided. Marks per section are shown in square brackets. Your answers should be either a Jupyter Notebook or Python code. Ensure that you comment your code, and give every function a docstring.*

1. (a) Consider the equation $z^3 - 1 = 0$, where $z = x + iy$. Write a function to implement the Newton-Raphson root finder for this function given an input of x_0 and y_0. [You do **not** need to use complex complex numbers: work with real and imaginary parts. However, **it is important to note that this is a one dimensional function**: you should only implement a 1D Newton-Raphson solver.] [6]

   (b) Now write loops to iterate over a square in the complex plane for $-1 \leq x \leq 1$ and $-1 \leq y \leq 1$ for an appropriate number of points. For each point, record one if the root returned is 1, and zero otherwise (one of the two complex roots). [3]

   (c) Plot the result using `imshow`. You may want to adjust the number of points you use in the square. If you want, you might experiment with returning the number of iterations, and colouring the plot in a similar style to the Mandelbrot set (no marks for this part). [1]

2. We will be exploring the following 2D function:

$$f(x, y) = \frac{xy}{100} \cos(2\pi x/5) \cos(2\pi y/3.333)$$

   (a) Create arrays for $-5.2 \leq x \leq 5.2$ and $-5.2 \leq y \leq 5.2$, and generate appropriate 2D versions using `np.meshgrid` [3]

   (b) Evaluate the function above, and plot using `imshow`. [2]

   (c) Write two python functions: the first, to calculate the function above; the second, to calculate its *vector* derivative (you will need to return an array with $\partial f/\partial x$ and $\partial f/\partial y$). [**NOTE**: your functions will need to take a numpy array (where `a[0]=x` and `a[1]=y` or similar) instead of (x, y).] [3]

   (d) Passing these two functions to `optimize.minimize` from `scipy`, find what result the conjugate gradient method returns for the minimum of the function (you need arguments `fun`, `x0`, `method='CG'` and `jac=dfun` where the python functions from the previous part replace `fun` and `dfun`). Be sure to try at least two starting points, and compare the result briefly to what you expect from looking at the form of the function and the plot. [2]

3. We will consider the function:

$$g(x, y) = \frac{3}{2}x^2 + 2xy + 3y^2 - x + 4y$$

   and explore its minimisation using steepest descents.

   (a) Write two python functions: the first to return $g(x, y)$ and the second to return its vector gradient. [Note: as in Q2 you will need to pass an array not two arguments.] [3]

   (b) For a starting point of $(x_0, y_0) = (5, -3)$, calculate the gradient using your second function. Pass *minus* the gradient as the search direction to the routine `optimize.line_search` from `scipy` (which takes arguments `f`, `fprime`, `x0`, `search` for the function, its derivative, the starting point and the search direction). The routine returns six arguments, the first of which is the optimum value of λ; if you store the return in a single variable, it will be an array (so if you set `alpha = optimize.line_search(...)` then the optimum value will be `alpha[0]`). What is the location of the minimum? [Note: do **NOT** create a variable called `lambda`, as this is a very specific keyword in python.] [2]

   (c) Now write a basic steepest descent function. It should take as input: a function; its derivative; a starting point; and a tolerance. You may use `optimize.line_search` if you wish. You should return the location of the minimum, the value at that point and the number of iterations required. [3]

   (d) Use your function to find the minimum value of $g(x, y)$ starting first at $(5, -3)$ and then at $(-3, -1)$. Note how the starting point affects the efficiency. [2]