Computational Physics, HW 2

Ben Johnston

September 2023

1 Problem 1

The purpose of this problem was to determine how NumPy's 32-bit floating point representation represents the number 100.98763 in bits and then calculate by how much this differs from the actual value. As seen in the script $Problem_1.py$ the floating point representation of this number in terms of the sign, exponent and mantissa is:

$$Sign = [0] \tag{1}$$

$$Exponent = [1, 0, 0, 0, 0, 1, 0, 1] \tag{2}$$

$$Mantissa = [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1] \tag{3}$$

The difference between the value represented here and 100.98763 was determined to be $2.75146484375 \times 10^{-6}$.

2 Problem 2

This question asked for the Madelung constant to be calculated for a 3D lattice of NaCl in two different ways: with for loops and without for loops. Labelling each position on the lattice by three integer coordinates (i, j, k), the sodium atoms fell on positions where $i + j + k$ was even and chlorine atoms fell on positions where $i + j + k$ was odd. The total potential felt by a sodium atom at $i = j = k = 0$ is the sum of the potential given by:

$$V_{total} = \sum_{i,j,k=-L}^{L} V(i, j, k) = \frac{e}{4\pi\epsilon_0 a} M \tag{4}$$

The script $Problem_2.py$ determines the Madelung constant in both of the prescribed ways, leading to a value of -1.7446850421707383 using for loops and -1.744685042168412 without. These values are the same to within 1×10^{-10} and agree well with the accepted experimental value.

3 Problem 3

The aim of this problem was to plot the Mandelbrot set, which is a fractal defined in terms of complex numbers through the following equation:

$$z' = z^2 + c \tag{5}$$

where z is a complex number and c is a complex constant. To determine the Mandelbrot set the above equation was repeatedly iterated with an initial value of $z = 0$ for a given constant c. If the magnitude $|z|$ of the resulting value is ever greater than 2 then the point in the complex plane is not part of the set. The script $Problem_3.py$ performs this iteration and then plots the Mandelbrot set, which can be seen below in Figure 1:

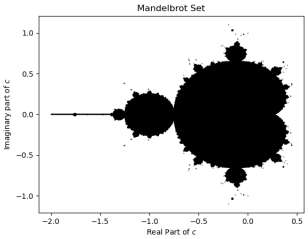

Figure 1: Mandelbrot Set

4 Problem 4

This problem involved writing a script that is able to solve a quadratic equation, more specifically the following equation:

$$0.001x^2 + 1000x + 0.001 = 0 \tag{6}$$

Code was first written to solve for x using the following equations:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{7}$$

which yielded values of $x_1 = -9.999894245993346 \times 10^{-7}$ and $x_2 = -999999.999999$. The same quadratic equation was then solved using the equation below:

$$x = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}} \tag{8}$$

The output of this equation yielded values of $x_1 = -1.000000000001 \times 10^{-6}$ and $x_2 = -1000010.5755125057$. A possible reason for the discrepancy between these two sets of values is that due to the difference in the orders of magnitude of the input values (i.e. 0.001 and 1000) so the accuracy of the floating point numbers will be different, leading to round off errors in the computation.

Following this, the problem asked for a script that accurately calculates the roots of both of a quadratic equation in all cases. Using the non-cancelling variant ($-b - \sqrt{b^2 - 4ac}$) and the fact that the product of the two roots should be equal to $-\frac{c}{a}$ a script was written that calculated the two roots of the quadratic equation accurately. In order to further check this, a unit test was performed on the script that was written and the script passed.