Computational Physics, HW 3

Ben Johnston

September 2023

1 Problem 1

The aim of this problem was to calculate the derivative of the function $f(x)$:

$$f(x) = x(x-1) \tag{1}$$

using the formula:

$$\frac{df}{dx} = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta} \tag{2}$$

evaluated at $x = 1$.

This was done computationally using values of $\delta = 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}, 10^{-12}, 10^{-14}$. Analytically, the value of this derivative is 1,l however in the computation we see the following:

Delta	**Computational Value of Derivative**
10^{-2}	1.100000000000001
10^{-4}	1.0009999999998895
10^{-6}	1.0000100000065513
10^{-8}	1.0000001005838672
10^{-10}	1.000000083740371
10^{-12}	1.000000082750371
10^{-14}	0.9992007221627407

Table 1: Derivative values

It can be seen here that initially the accuracy of the calculation gets better as δ decreases, however at $\delta = 10^{-14}$ the accuracy decreases again. The reason for this could be that this part of the computation involves the subtraction of two numbers that are very nearly equal (and comparable to the accuracy of the computer), which could result in a large fractional error.

2 Problem 2

This problem involved calculating empirically how matrix multiplication computation increases with matrix size. This was computed using an explicit function using for loops and using NumPy's dot() method.

Considering first the for loop method, we see the following relation between matrix size and multiplication computation time:

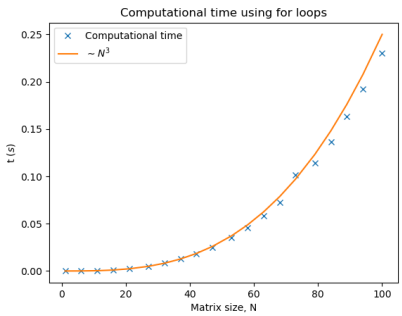

Figure 1: For loop computation time

It can be seen here that for this method the computation time increases as N^3 as predicted. This is due to the three nested loops within the function defined in the script *Problem*$_2$*.py*.

Now considering NumPy's dot() method we see a different relation between computation time and matrix size:

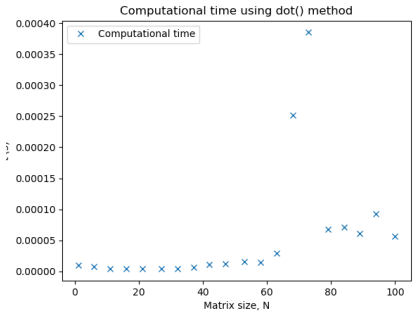

Figure 2: dot() method computation time

From this graph it is clear to see that the dot() method does not follow the predicted N^3 relation. This is probably due to the fact that this NumPy method is a much more efficient method to computationally multiply matrices together.

3 Problem 3

The aim of this problem was to computationally model the radioactive decay of the unstable bismuth isotope ^{213}Bi to the stable bismuth isotope ^{209}Bi via two different routes (as seen in exercise 10.2 of Newman). Starting with a sample consisting of 10,000 atoms of ^{213}Bi the decay was modelled by dividing time into slices of $\delta t = 1$, and *Figure 3* below shows this decay:

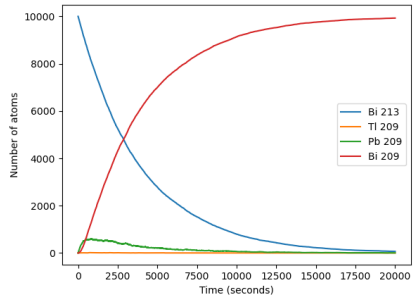

Figure 3: Radioactive Decay of ^{213}Bi

From this it can be seen that as a function of time, the amount of ^{213}Bi decreases in a similar way to the amount of ^{209}Bi increases. The reason for the slower initial increase in ^{209}Bi is due to the increase and decrease of ^{209}Pb and ^{209}Tl before they eventually decay into ^{209}Bi.

4 Problem 4

This problem involved computing the decay of the radioactive isotope ^{208}Tl to the stable ^{208}Pb using the transformation method described in Newman. This essentially took an array of uniformly distributed random numbers, z and transformed them into exponentially distributed values, x using the equation:

$$x = -\frac{1}{\mu}\ln(1-z) \tag{3}$$

where:

$$\mu = \frac{\ln 2}{\tau} \tag{4}$$

These x values were then sorted and plotted to show the number of atoms not decayed as a function of time, as seen below:

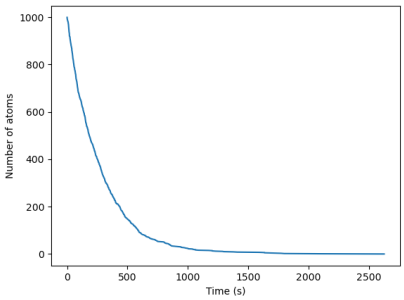

Figure 4: Radioactive Decay in ^{208}Tl

From this it can be seen that the expected exponential decrease in the number of ^{208}Tl atoms is observed.