[PHYS-GA2000] Problem Set 3

Dylan Lane
Github: dl4729

September 26, 2023

Problem 1

Let $f(x) = x(x-1)$; analytically, this has derivative $f'(x) = 2x-1$, and it follows that $f'(1) = 1$. This is the result to which the numerical differentiations will be compared to. By the definition of the derivative, $f'(x) = \frac{f(x+\delta)-f(x)}{\delta}$ as δ is taken to 0, and although the computer cannot take the continuous limit it can approximate progressively smaller δ. Construct an array $\delta = [10^{-2}, 10^{-4}, ..., 10^{-14}]$; then take $f(x) = x(x-1)$ at $x = 1+\delta$, subtract $f(1)$ and divide by the array δ to get an array of numerical derivatives taken at the various values of δ. Taking log base 10 scaling to assess how the magnitude of the error changes, observe Figure 1.

As described in the textbook [1], the error begins by decreasing as δ gets smaller (which corresponds to the +x axis in Figure 1), then beginning to increase around $\delta = 10^{-8}$. This is likely due to roundoff error; if δ gets any smaller than this, the multiplication and subtraction in the calculation of $f(x)$ begins to suffer from information loss due to the limitations of floating point numbers on a computer. Consequently the error increases and continues to get worse as one gets smaller and smaller δ.

Problem 2

For both methods of implementing matrix multiplication (for loops and NumPy's dot function), the setup is trivial. For the for loops, simply create two matrices A and B of zeros and iterate through them, populating a new matrix element by element with a loop over the indices of a particular row and column of A and B respectively [1]. To show the relationship between N the number of rows/columns of the matrices, iterate over N (in my case for N=10, 30, 50, and so on up to 190) and time each matrix multiplication. This generates Figure 2.

It is then evident that as expected, $t \propto N^3$.

The dot method is even more straightforward as it can simply be applied to two matrices and output the product. Timing the multiplication for each N as in the for loop, this relationship seems more random than in the case of the for loops. The plot in Figure 3 is from one particular run, but the features like the position of the peak vary across runs; it seems likely to me that other computer processes taking place concurrently to the run are bigger factors than the size of the matrices being multiplied in this computation.

The plots are overlaid in figure (4) with linear scaling in x:

Problem 3

From the textbook, we know that for a given decay process with half-life τ, the probability of a particle decaying in a particular timestep is given by

$$p = 1 - 2^{-dt/\tau} \tag{1}$$

which follows from the decay equation $N = N_0 * 2^{t/\tau}$ [1]. To simulate the decay of a particular set of nuclei to another, we must simply pick a random number from a uniform distribution between 0 and 1 for each nucleus and if this number is less than the probability of decaying, reclassify which nucleus it is. The more complicated decay chain in the problem can be broken down into its constituent decay processes, each with different parameter τ_i and applied consecutively, starting from the end so it is impossible for a particle to decay twice in a timestep. To summarize, the entirety of the algorithm is to maintain one variable for the count of each element involved in the process (Bi-213, Tl-209, Pb-209, and Bi-209) and update these after each time step in which a certain number of nuclei by the picking of a random number.

The complication is that Bi-213 has two potential decay products (Tl-209 and Pb-209) and which one a single Bi-213 nucleus will decay into is determined probabilistically. The problem states the chance that it will decay into $Pb-209$ is about 97.91%. To account for this, for a given time step we choose a random number for each nucleus. Then, if the random number is small enough that the nucleus will decay, we draw another random number from a uniform distribution on $[0, 1)$; if it is less than .9791, it will decay into Pb-209, otherwise it will decay into Tl-209.

Figure 5 is generated from running this process over 20000 seconds with a step length of 1 second and from a starting point of 10000 Bi-213 atoms and nothing else. The result is as expected; the number of Pb-209 and Tl-209 atoms present at a particular time is low because they decay much more frequently than the Bi-213 which generates more of them. In 20000 seconds there are over 7 half-lives of Bi-213 which occur, so the final number of Bi-213 should be order 10 (approximately 70 molecules left); checking the final number of atoms, this is indeed approximately what is observed.

Problem 4

This amounts to reformulating the decay problem as a selection of random numbers from a nonuniform distribution, namely $P(t)dt = 2^{-t/\tau}\frac{ln(2)}{\tau} = \mu e^{-\mu x}$

for $\mu = ln(2)/\tau$. By integrating to generate an exponential distribution as described in the textbook, we find $z = 1-e^{-\mu x}$ where z is a uniformly distributed variable. Then $x = -\frac{1}{\mu}ln(1-z)$ [1]. To generate x from the desired distribution is suffices to generate z from a uniform distribution on $[0,1)$ and substitute the uniform numbers into this map from z to x for an array of numbers randomly chosen on the exponential distribution. Setting the proper value of τ, this assigns each nucleus in our set a decay time; at each chosen time, the number of nuclei will decrease by 1 (thus the y-axis is simply given in Python by 1000 minus an array from 1 to 1000 inclusive. The resulting plot is shown in Figure 6.

Figures

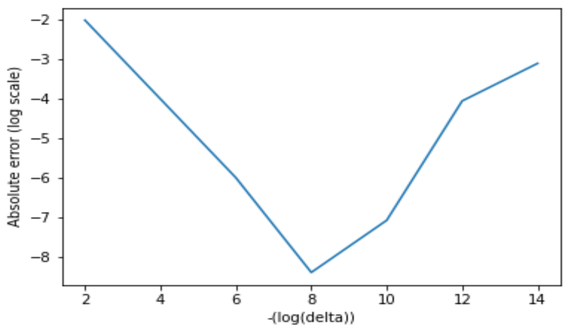

Figure 1: (Problem 1) Errors in the Numerical Derivative with different δ

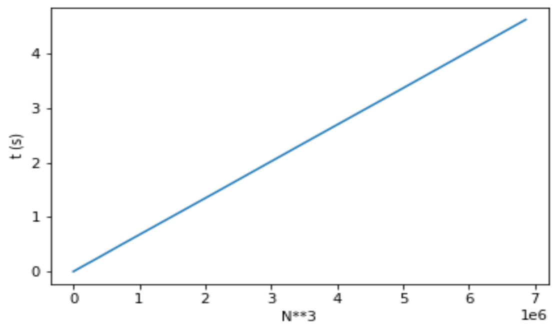

Figure 2: (Problem 2) Plot N^3 (N number of rows/columns in the matrices being multiplied) vs. t

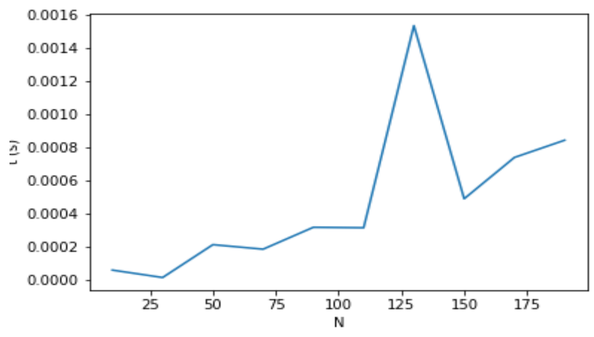

Figure 3: (Problem 2) N vs. t (s) using NumPy dot function.

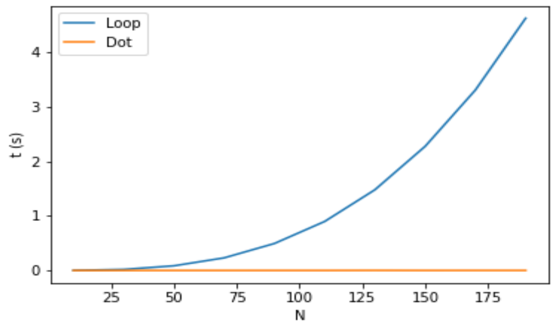

Figure 4: (Problem 2) N vs t(s) with both methods displayed

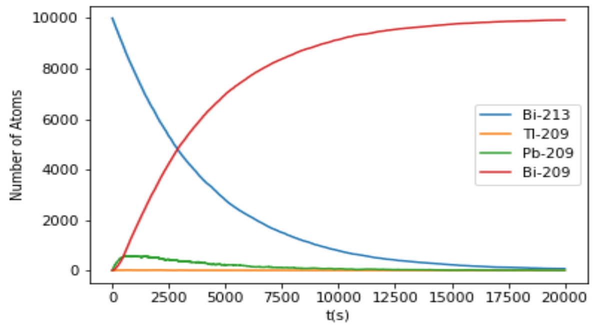

Figure 5: (Problem 3) Time (seconds) vs. Number of Atoms over 20000 timesteps

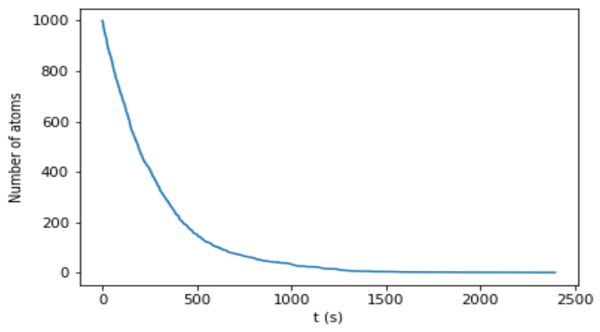

Figure 6: (Problem 4) Time (seconds) vs Number of Atoms

References

[1] Newman, M. 2012, Computational Physics (Createspace Independent Pub)

[2] `https://blanton144.github.io/computational-grad/`

Also discussed results with Nancy Shi.