Problem Set 3

Evan Arch (https://github.com/evan-nyu/phys-ga2000)

September 25, 2024

Problem 1

Multiplying NxN matrices in the naive way, using a triple for loop, has a theoretical runtime of $\mathcal{O}(N^3)$. By testing the multiplication on $N \times N$ matrices ranging in size from 10×10 to 200×200 and plotting the results, I have verified the $\mathcal{O}(\mathcal{N}^{\ni})$ runtime of the method and compared it to numpy's built in dot() method (see figure 1). Numpy's dot method for matrix multiplication appear to also operate in $\mathcal{O}(N^3)$ time, however, it is still faster than the for loop method in all tested cases.

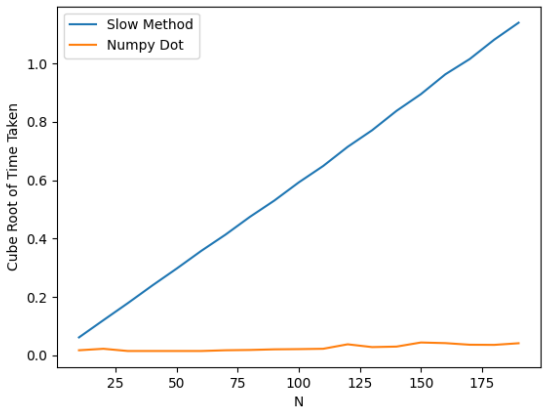

Figure 1: Plot of the cube root of the runtime in seconds versus the matrix size for the for loop method (blue), and numpy's dot method (orange).

Problem 2

A simple method for simulating the decay of an is to compute the probability p that an atom will decay in one second,

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

where τ is the half-life, pick a random number between zero and one, then compare the probability of decay to the random number. If random number is less than the probability, the atom has decayed in that second. This can be repeated across a large population of atoms and seconds to generate behavior that mimics radioactive decay.

213 Bismuth has multiple decay paths (see figure 2), this adds an extra factor that must be considered during decay, what will the result be. Another random check is used to deal with this case. Once a decay has occurred, pick a random number, if the number is less than the percentage of the probability for product A to be the decay product, then the result is product A. Otherwise it is product B.

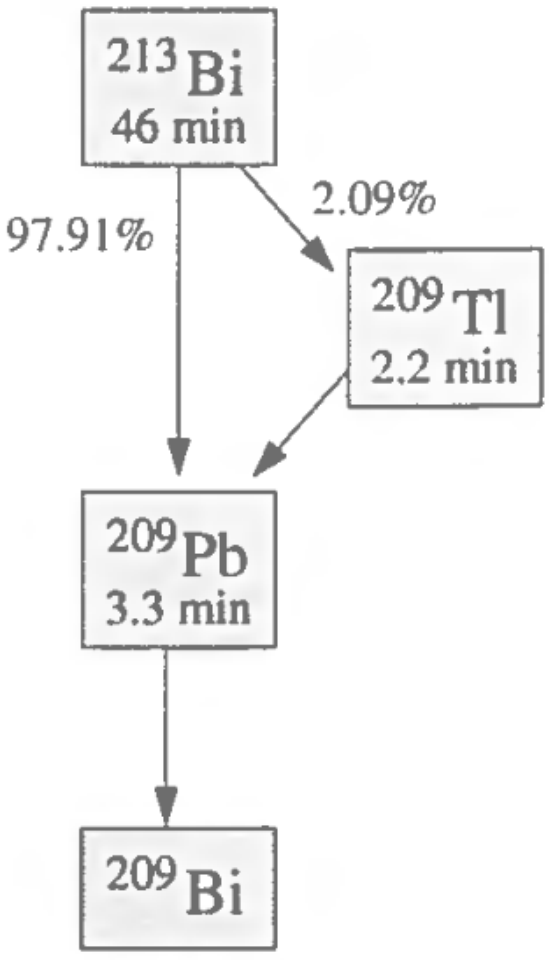

Figure 2: Decay Paths of ^{213}Bi.

Implementing 213 Bismuth's decay path in code is a task that can get complicated quickly. To simplify the code and keep it organized, I choose to represent each isotope as its own instance of an isotope class. Each class contains all the information related to a given isotope, population, half-life, decay products, etc. This allowed me to simply call an isotope's decay method to have it check its entire population for decays, and add said decays to the population of its result. Importantly, decays are done from the bottom up, starting with 209 Bismuth and ending at 213 Bismuth to prevent an atom from decaying twice in a single time step.

At the end of each time step the populations of each element are recorded. A plot of said populations as a function of time is shown in figure 3. The simulation was started with 10000 213 Bismuth atoms and ran for 20000 seconds.

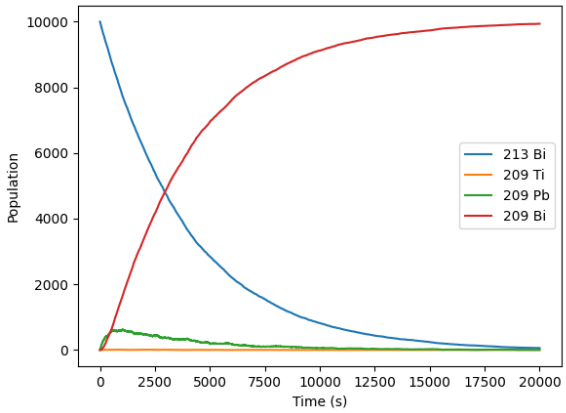

Figure 3: Populations of each isotope in ^{213}Bi's decay path as a function of time.

Problem 3

An easier way to simulate the decay of atoms is to first generate a list containing the times at which each atom will decay. This can be done by generating random numbers from a random distribution that matches the decay distribution. As radioactive decay follows an exponential distribution, we can use equation 10.10 from Newman

$$t = -\frac{1}{\mu}\log(1-z) \tag{2}$$

where t is a random variable with an exponential distribution, z is a random number with even distribution between 0 and 1, and $\mu = \log(2)\tau$ where τ is the half life in seconds. The random variable t is interpreted as the time at which an atom will decay. Generating a list of 1000 t, one can pick a time t_0, and the number of ts where $t > t_0$ is the number of atoms the remain undecayed. Stepping slowly incrementing t_0 allows the generation of a population vs time plot (see figure 4)

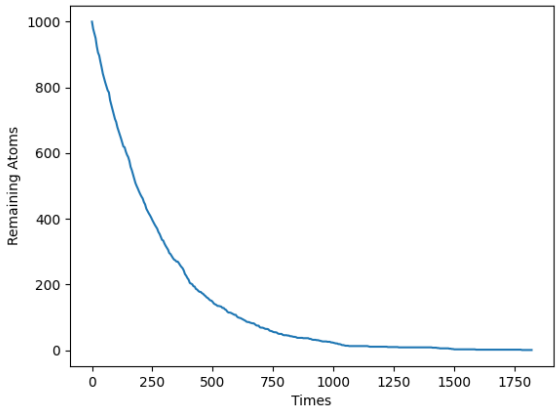

Figure 4: Plot of remaining $^{2}08$Ti atoms with $\tau = 3.053$ minutes versus time in seconds.

Problem 4

We are demonstrating the central limit theorem by generating the random variate y which depends on the random x which follow on exponential distribution,

$$y = \frac{1}{N}\sum_{i=0}^{N} x_i \tag{3}$$

where N is an integer. If we take a random distribution of y values, we expect that the mean will remain constant at 1 since

$$\bar{y} = \frac{1}{N}\sum_{i=0}^{N} \bar{x_i} = \bar{x}, \tag{4}$$

and the mean $\bar{x}$ equals one since $\bar{x} = \frac{1}{\lambda}$ and we set $\lambda = 1$. We expect that the variance of a distribution of ys will vary with $\frac{1}{N}$ since

$$\mathrm{Var}(y) = \mathrm{Var}\left(\frac{1}{N}\sum_{i=0}^{N} x_i\right) = \frac{1}{N^2}\left(\mathrm{Var}(x_1) + \mathrm{Var}(x_2) + ...\mathrm{Var}(x_N)\right) = \frac{1}{N}\mathrm{Var}(x) \tag{5}$$

and $\mathrm{Var}(x) = 1$ since $\lambda = 1$.

Plotting a normalized distribution of 1000 ys with $N = 1000$, we see that the distribution of ys becomes a Gaussian (see figure 5).

Using numpy and scipy's built in functions, we can easily calculate how the mean, variance, skewness, and kurtosis of the distribution of ys vary with increasing N (see figure **??**). The skewness first reaches 0.01 of its original value at $N = 725$. The kurtosis first reaches 0.01 of its original value when $N = 49$. These test were run with varying N in increments of 2 from 1 to 1000 and with 10000 ys in each distribution.

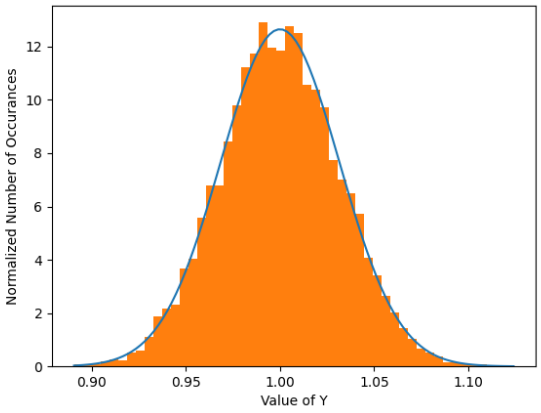

Figure 5: Normalized histogram of a distribution of 10000 ys with $N = 1000$. A normalized Gaussian is plotted over the top to show agreement

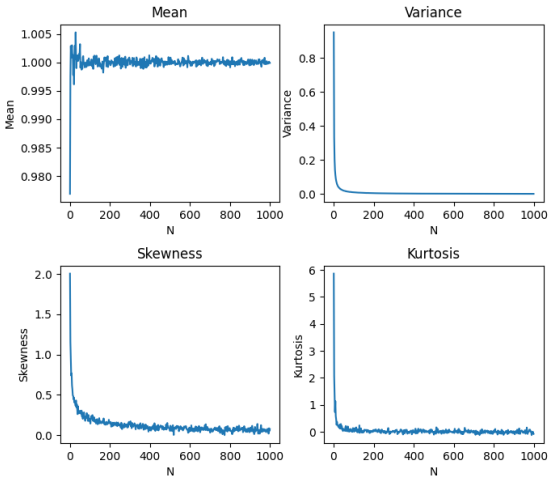

Figure 6: Plots of mean, variance, skewness, and kurtosis as a function of N.