PHYSICS 20323: Scientific Analysis & Modeling - Fall 2023
Project: Ahabar Hossain

PROJECT INFORMATION:

Final Project for Scientific Analysis and Modeling

PURPOSE:

The purpose of this project is to study a radioactive decay process and to model the decay and energy resulting from the process using python. Through our modeling, we will statistically analyze the results from our code in order to answer questions related to the decay energies.

PROCEDURE:

The following procedure will give a brief outline to the code we have established. A more detailed look at the code can be seen in the actual code created through python.

1) Modeling the radioactive process

Step 1 Creating Variables:

In order to model the radioactive decay process, we will first set in import the following into our code: ”**numpy as np from numpy import arange from pylab import plot,xlabel,ylabel,show import random”**. From here, you will essentially create variables that hold the number of atoms for each of the elements. For this project, set all of these variables = 0 except the atom first in your decay process, which will have the number of atoms you start with (20,000 in this case). Then create variables for the half lives of each element; this will help determine the probability factor in your decay processes later on. Your half life variables should be in seconds, so if your instructor gave you a half life in minutes, then you can multiply it by 60 in order to put it in seconds. From here, we will now create variables for the probability decay for each element. Do this by using the formula p(t) = 1 - 2t/half life. Your t here in this case will be an h = 1. After this, set a tmax value for the x-axis of your eventual (adjust this when you see your plot accordingly. For our case it was 3000 seconds). After this, do tpoints = arange(0.0,tmax,h) - you will use this in your for loop. Create ”points” arrays such as, Atpoints = [], for each of your elements respectively.

Step 2 Creating For Loops for each element to track decay processes:

Consequently, we can now proceed to create the for loop that will actually track the decay throughout the process. The rule of thumb will be to start from bottom to up in the decay process in order to avoid making the same atom inadvertently decay twice in the same step. In order to start the for loop, do for t in tpoints, In order to proceed, append all the original variables you created to hold the number of atoms in that element to the array you created for that element respectively. In order to track the decay process for each step in the radioactive decay process, work from the bottom up and basically trace the arrow backwards to the element it came from. First, create a decay variable like decay1 for the first step and set it equal to 0. Set a for loop in the range of this element using the number variable for the element you created in the beginning. Within your for loop, create an if statement using the np.random.random() function ¡ the probability for that element. Add 1 to your decay variable, subtract from the element you started with in the for loop with this decay variable, and add this decay variable to the element the arrow points toward. If the part of the process you are coding for has an additional probability percentage, such as 95 percent chance it goes one way and 5 percent chance it goes the other way, then instead of adding and subtracting to your atoms right away, simply add to your decay variable first, create another if statement within the current if statement using the same random function and set it ¡= the lowest percentage. Underneath this statement, subtract one from the element you start with and add 1 to the element you end up with. Then, do else and repeat this for the element the process goes to in all the other chances of time. After you have done all of this for one of the parts of the decay process, keep doing this for all of the atoms respectively until you reach the beginning of the decay process.

Step 3: Plotting the decay process in a graph:

Now that you have done this, you can plot the number of times over time in a graph by doing plot(tpoints, *the respective array you created for the element, color). Do this for all of the elements and label your x and y axes ”time” and ”number of atoms” respectively. Finally code the command show().

2) Calculating Number and Energy generated from decay processes:

In this section, getting the amount of times for each process and the energy generated from should be pretty simple. Create a new cell, and copy and paste the code you just created. First, close to where you put the number variables in your first line of coding, create additional variables for both the energy and number of energies and set them equal to zero. For example, for beta process, you can do betaenergy = 0 to track the number of beta energies and beta = 0 to track the number of beta processes taking place during the process. Once you do this for all the elements, go back to the radioactive decay process diagram, see which arrows correlate with each decay process and how it relates to the elements decaying or not. Now go back to your code and look at your for loops. In the for loops that correspond to the elements you saw decaying and to the other element where the arrow was going towards, simply put your energy and number variables underneath the respective if statement and insert a += 1 to the variables. Once you have done these for all the decay processes according to their respective sports in the diagram, create two variables, one for the summ of all the energy processes, and another for the total sum of all the processes. After this, now you can simply do print statements by making each of the variables a string to see your results.

3) Average and Standard Deviation of total and Individual Energies:

In order to calculate the average and standard deviation of the total and individual energies, first copy and paste the code you have created into another cell. Then since you need to have a minimum of 10 runs, create a variable with x = 10, and create arrays for your decay energies close to where you created your original variables. After that, highlight your entire current for loop for t points and shift it to the right once with the tab command. This for loop will now be within your new for loop which will essentially iterate your program 10 times: for i in range(x): . Now, within this for loop and in line with the original for loop, append your variables that counted each of the decay processes to your array variables that you created like this : arrayvariable = np.append(arrayvariable, energyvariable). Once you have done this, you can now proceed to the print statements. Firstly, you want to show the energy generated for each run, so create a print statement like this: print(”The beta energy of run number” + str(i+1) + ” is ”+str(average10runsbeta[i])). Do this for all your energies using their respective array variables. Now, in order to find the individual averages of the energies simply use a print statement that takes the average of your array variable using np.average, and do the same thing for standard deviation using np.std . Make sure to turn your average and standard deviation values as a string. Finally, in order to take the total average and the total standard deviation of all energies, create variables for the total average and standard deviation where each variable adds up the averages and standard deviations of the decay processes respectively. Take the string of these variables and put them in their respective print statements to see your results.

ANALYSIS (and math):

Question: As alpha particle decay is very harmful to humans, how thick (cm) of a shield would be needed to safely block ALL alpha-particle decay energy, if each 1 cm blocks = 3,500 MeV.

Since, according to our results below, the total average of alpha energies = 32785 MeV, the 3 standard deviation of alpha energies would mean 9 because our results show a standard deviation of 3 for alpha energies, then:

$(32785meV + 9meV) \div 3,500meV/cm = 9.370cm$

So, 9.370 cm thick of shield would be needed to safely block all alpha particle decay energy.

RESULTS:

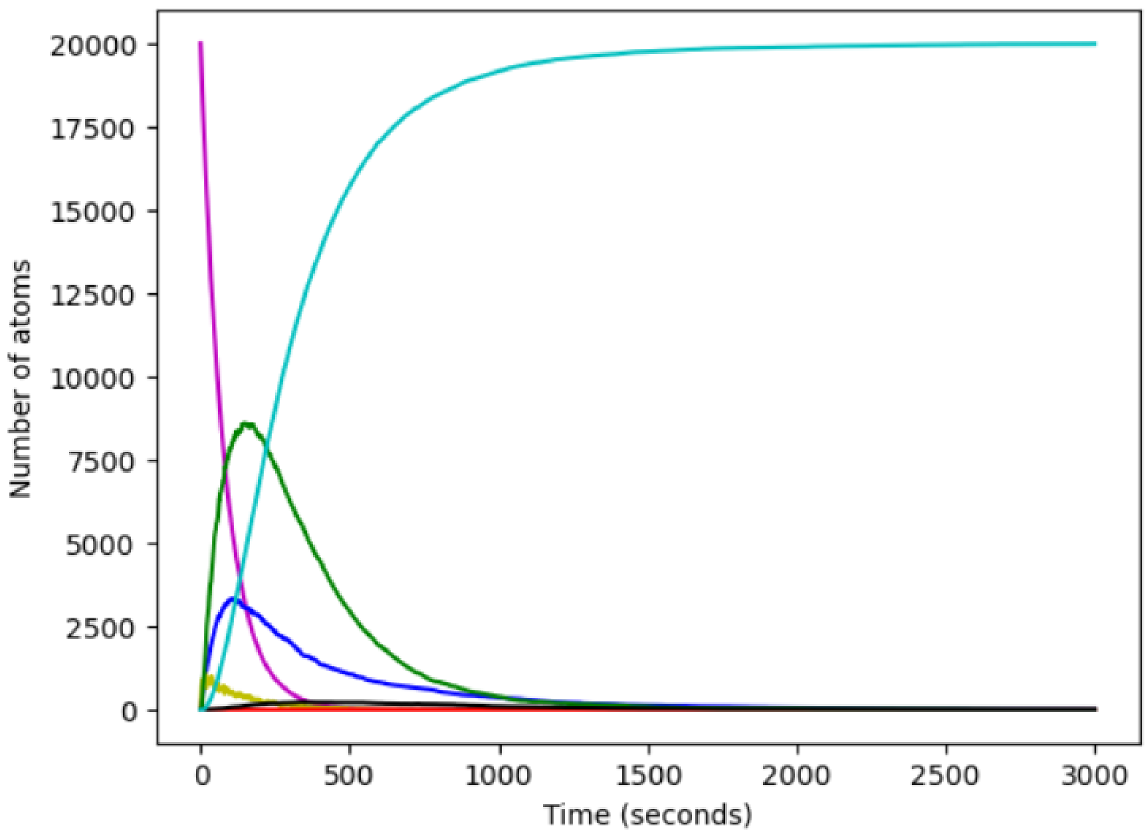

Figure 1: Modeling Radioactive Decay Process; Colors indicate elements shown in jupyter notebook

Beta Energy (MeV)	Alpha Energy (MeV)	R Energy (MeV)	Z Energy (MeV)
36237.5	33055	153139	23004

Table 1: Total beta energies from 10 runs. Sum of all Energies: 245435.5 MeV

Beta Processes	Alpha Processes	R Energy Processes	Z Energy Processes
14495	6611	2556	21877

Table 2: Each of total Number of Energy Processes for 10 runs. Sum of all Energies: 45539

Runs	Beta Energy (MeV)	Alpha Energy (MeV)	R Energy (MeV)	Z Energy (MeV)
1	36482.5	32775	153167	23256
2	36490	32785	153384	23256
3	36490	32785	153384	23256
4	36490	32785	153384	23256
5	36490	32785	153384	23256
6	36490	32785	153384	23256
7	36490	32785	153384	23256
8	36490	32785	153384	23256
9	36490	32785	153384	23256
10	36490	32785	153384	23256
Average	36489.25	32784	153362.3	23256
Standard Deviation	2.25	3.0	65.1	0

Table 3: Energies through 10 runs: Total Average of Energies: 245891.55 MeV, Total Standard Deviation of Energies: 70.35

CONCLUSION:

In conclusion, we described how to model a radioactive decay process consisting of multiple different decay energies: alpha, beta, R, and Z. We statistically analyzed our data to conclude that 9.37 cm o f a thick shield would be needed to block all alpha particle decay energy according to our results. From our results, we found that the total beta, alpha, R, and Z energies after 10 runs were 36237.5, 33055, 153139, and 23004 MeV respectively. We also found that after 10 runs, the total number of beta, alpha, R, and Z processes were 14495, 6611, 2556, 21877. In addition, we discovered that the total average of the beta, alpha, R, and Z eneriges were 36489.25, 32784, 153362.3, and 23256 MeV respectively. Finally, the standard deviations for beta, alpha, R, and Z energies were 2.25, 3.0, 65.1 and 0 respectively.