Radioactive Decay using Monte Carlo Method

Bijin Joseph (182361). Hetanshu Bharadiya (182343)

26th March 2021

Abstract

In this Project we analyse the radioactive decay of an element using the Monte Carlo method. We have first consider a single element's decay and later on have simulated successive decay of ^{225}Ra. We have also developed our code to obtained the average number of particle decay in the first iteration (ie first unit of time) and have observed the Poisson's curve for it.

Monte Carlo method, Successive Radioactive Decay, ^{225}Ra, ^{225}Ac, Poisson's curve

1 Introduction

Starting with the discovery of X-rays by Roentgen and later on developed by the french scientist Becquerel, Radioactivity was first observed in Uranium. Later on Marie Curie and her husband, Discovered two new elements, polonium and radium which emitted the becquerel rays.
Radioactivity is the phenomenon exhibited by the nuclei of an atom as a result of nuclear instability and it is the property possessed by some elements (such as uranium) or isotopes (such as carbon 14) of spontaneously emitting energetic particles (such as electrons or alpha particles) by the disintegration of their atomic nuclei. Radioactive decay, also known as nuclear decay, radioactivity, radioactive disintegration or nuclear disintegration, is the process by which an unstable atomic nucleus loses energy and a material containing unstable nuclei is considered radioactive. Except for gamma decay or internal conversion from a nuclear excited state, the decay is a nuclear transmutation resulting in a daughter containing a different number of protons or neutrons (or both).
Monte Carlo method are computational algorithms that rely on repeated random sampling to obtain numerical results. It uses the concept of randomness to obtain deterministic result and a classic example of random behavior is nuclear decay. Each radioactive isotope has a certain intrinsic probability of decaying per unit time. As far as we can tell, the time when any particular nucleus decays is truly random. Therefore Monte Carlo method is an integral part of our code.

2 Discussion

2.1 Theory

In Radioactive decay, The rate of decay of radioactive material is measured by the lifetime τ. Consider $N(0)$ atoms at time $t = 0$, the number of atoms remaining after time t is given by

$$N(t) = N(0) \exp(-t/\tau) \,.$$

and $t_{1/2}$, known as the half time of decay is the time it takes for half of the atoms to decay. It can be found from the formula:

$$t_{1/2} = \log(2)\tau \,.$$

The decay rate is the number of decays per unit time at any instant of time and is given by the equation

$$dF(t)/dt = N(t)/\tau$$

If we observe the decay over a time dt, that is very short as compared to the life time of the particle, the number of atoms stay constant to an approximation. However, there will be a small decay and this number of decay over that time period will be

$$d = \frac{dF}{dt} dt = N(0) \frac{dt}{\tau} \exp(-t/\tau) = \frac{dt}{\tau} N(0) \,,$$

In practice, because of fluctuations, there may be more or fewer decays. The probability of observing k decays is given by the Poisson distribution:

$$P(k) = \frac{d^k}{k!} \exp(-d)$$

where $k!$ is the factorial $k(k-1)(k-2)\ldots 1$. If we make M measurements of the decay over a total time period Mdt, then the number of intervals dt in which exactly k decays occur is given by

$$N(k) = MP(k) = M \frac{d^k}{k!} \exp(-d) \,.$$

The number of nuclei of a given radioactive sample, disintegrating per second is called the activity of that sample so

$$\frac{dN(k)}{dt}$$

= Rate of decrease of nuclei with time = Activity at time t

The activity at a particular time is proportional to the number of parent atoms.

Decay constant, being proportionality between the size of a population of radioactive atoms and the rate at which the population decreases because of radioactive decay, is given by the equation, = 0.693/T1/2.
Consider decay type of

$$A \rightarrow B \rightarrow C(stable)$$

Here, the maximum activity of particle B, will follow the equation

$$t_{max} = \frac{log(\lambda_B/\lambda_A)}{\lambda_B - \lambda_A}$$

where λ_A and λ_B are the decay constant of A and B respectively.

For our experiment we will be considering 225 Radium, decaying to 225 Actinium. The reason why we took these elements is due to their close half lives. For 225 Radium, $t_{1/2}$ is 14.9 days and that of 225 Actinium is 9.92 days.

CODE

Code for Simulating a single decay

```
#for importing the required packages
import numpy as np
import random as rn
import matplotlib.pyplot as plt
import math as m

#definition to count the number of zeros
def zeros(q):
    n = 0
    for i in range(len(q)):
        if q[i]==0:
            n += 1
    return(n)

n = 100000 #number of particles
p = 0.001 #probability of each decay
q = np.ones(n) #particles given each matrix location
L = []  #to find the number of decay number
itera = 0

while N < 0.9*n: #going over the all the particles until 90% decay is obtained
    itera += 1
    for i in range(n): #going through each particles
        if q[i]!=0:    #checking if the particle has already decayed
            if rn.uniform(0,1)< p: #checking if the probability is statisfied
                q[i] = 0         #if yes then it decays (indicated by zero)
    N = zeros(q)  #calculating the number of zeros (decays)
    L.append(n-N) #storing them
print(itera)

T = np.arange(itera)
plt.plot(T, L, label = "Monte carlo method")
plt.title(n, 'particles')
plt.plot(T,n * np.exp (-T*p),linestyle = '--', label = 'Ideal')
print(np.log(2)/p)
plt.legend(loc='upper right')
plt.show()
```

Code for obtaining the poisson's curve

```
#importing the required packages
import numpy as np
import random as rn
import matplotlib.pyplot as plt
import math as m

#definition for calculating the number of decay
def zeros(q):
    n = 0
    for i in range(len(q)):
        if q[i]==0:
            n += 1
    return(n)

nl =[1000,5000, 10000, 50000, 100000] #set of number of particles
p = 0.001 #probability

#N = (zeros(q))

itera = 0
for n in nl:
 L = []
 q = np.ones(n)
 for j in range(1000):
    q = np.ones(n)
    for i in range(n):
        if q[i]!=0:
            if rn.uniform(0,1)< p:
                q[i] = 0
    N = zeros(q)
    L.append(N)
 plt.figure()
 plt.hist(L, density = True)
 plt.title(n)
 plt.show()
```

Code for Successive Decay

```
#importing the required packages
import numpy as np
import random as rn
import matplotlib.pyplot as plt
import math as m
#definition to calculate number of decays
def zero(q, s):
    n = 0
    for i in range(len(q)):
        if q[i]==s:
            n += 1
    return(n)
```

```
#defintion to find an element in a matrix
def find(element, matrix):
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if matrix[i][j] == element:
                return (i, j)

t_half_a = 14.9 #half life of Radium
t_half_b = 9.92 #half lifeof Ac
timepoint = 50  #number of time steps
t1 = 200        #total time
dt  = t1 / timepoint #time step
n = 10000 #number of particles
pa = 1 - np.exp(-dt / t_half_a * np.log(2)) #Calculating the decay probabilities
pb = 1 - np.exp(-dt / t_half_b * np.log(2))
qa = np.zeros(timepoint) #number of Ra decay in each time step
qb = np.zeros(timepoint) #number of Ac decay in each time step
dqa = np.zeros(timepoint) #Change in number of Ra decay in each time step
dqb = np.zeros(timepoint) #Change in number of Ac decay in each time step
q = np.ones((n))          #matrix of each particle

#iterating till the time reaches
for j in range(timepoint):
    qa[j] = zero(q, 1) #to find the number of decays
    qb[j] = zero(q, 2)
    if j> 1:
        dqa[j] = qa[j] - qa[j-1] #calculating the change
        dqb[j] = qb[j]-qb[j-1]
    #going through each particle
    for i in range(n):
        #checking if a particles has not decayed at all
        if q[i]==1:
            if rn.uniform(0,1)< pa: #checking if the probability is statisfied
                q[i] = 2 #then decay to Ac
            else:
                q[i] = 1 #else will it wont
        elif q[i] == 2:  #checking if Ra has decayed to Ac
            if rn.uniform(0,1)< pb: #checking if the probability is satisfied
                q[i] = 3  #then decay
            else:
                q[i] = 2 #else not



timebase = np.arange(0, t1, t1/timepoint)
plt.plot(timebase, qa, label = 'Radium')
plt.plot(timebase, qb, label = 'Acitinium')
print(pa)
plt.figure()
plt.plot(timebase,dqa)
plt.plot(timebase,dqb)
val =max(dqb)
print(dqb.index(val))
```

2.2 Result

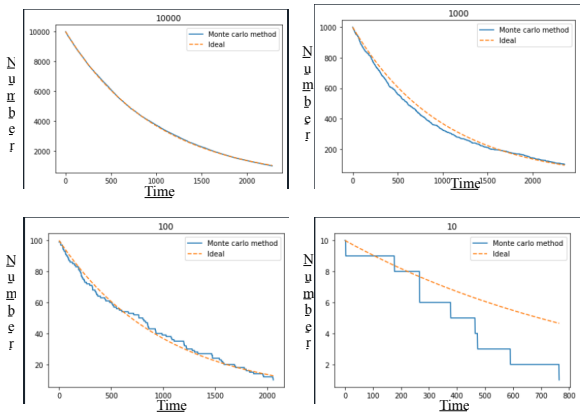

Figure 1: **Results of the code for Simulating a single decay**

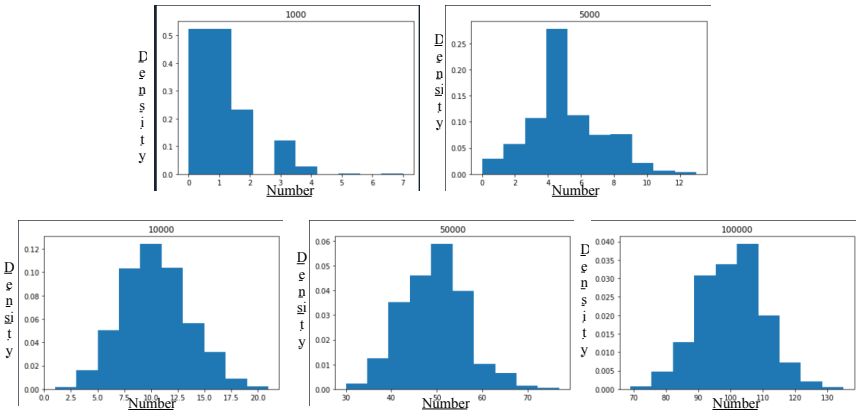

Figure 2: **Results of the code for obtaining the Poisson's curve for probability 0.001**

- In figure 1, we can see how the programs prediction increases with the number of particles. For initial number of particles of 10,000, the simulation gives the actual result. For n= 10000 the predicted half life is 696, for n = 5000 the predicted half life is 697, for n= 100 the predicted half life is 671, for n= 10 the predicted half life is 714. The theoretical value for half life is 693.14
- In figure 2 - 4, we can observe that the curve follows a Poisson's equation. We can also see how the curve varies for different probabilities.

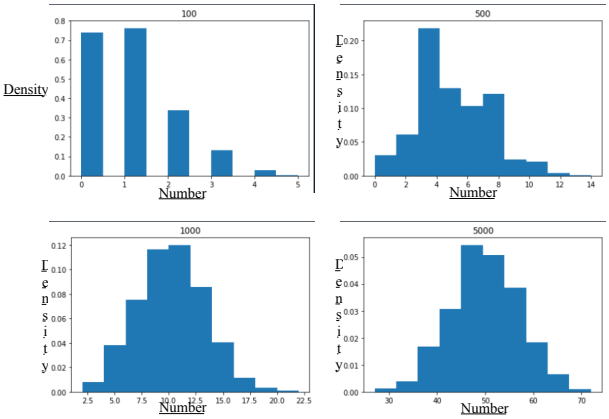

Figure 3: **Results of the code for obtaining the Poisson's curve for probability 0.01**

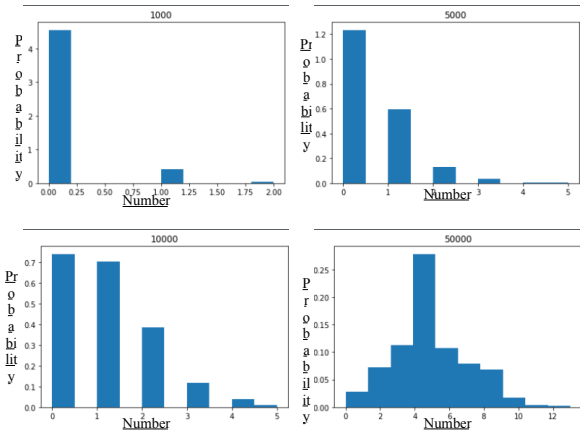

Figure 4: **Results of the code for obtaining the Poisson's curve for probability 0.0001**

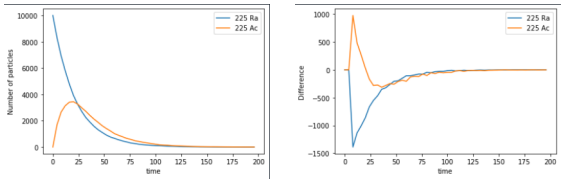

Figure 5: **Code for Successive Decay of 225 Ra**

- In figure 5, here we can observe how the decay for the successive decay of 225 Ra and 225 Ac, varies. We have also plotted the activity of the decay and we can see that the maximum time occurs at a time of 8 days, were as theoretically the value is 7.57

3 Conclusion

From this above experiment we have proved that Monte Carlo method can be used simulate radioactive decay. We have compared our results with theoretical values and the observations where similar for large values of N.

References

[1] S.B Patel "Introduction to Nuclear Physics"

[2] Werner Krauth "Statistical Mechanics: Algorithms and Computations"

[3] Daniel V. Schroeder "Physics Simulations in Python"

Appendix A For 99 Mo and 99m Tc