Himesh A M
BSC Physics
Roll no: 67

LAB ASSIGNMENT

MONTE - CARLO SIMULATION OF ~~RADIOACTV~~ RADIOACTIVE DECAY

AIM

- Implement a simulation of radioactive decay in python
- Provide initial conditions (number of particles, decay constant and analyse the results, including plotting the decay curve over time)
- Calculate the half-life of the radioactive substance based on the simulation results and check how it compares to the theoretically expected half-life

Provide information about a specific radioactive isotope with a known half-life to simulate the decay of this isotope and compare the simulation results with the expected decay

PRINCIPLE

The decay of an unstable nucleus is entirely random in time so it is impossible to predict when a particular atom will decay. But the number of decay events dN expected to in a a small interval of time dt is proportional to the number of atoms present N, given by the equation

$$\frac{dN}{dt} = -\lambda N$$

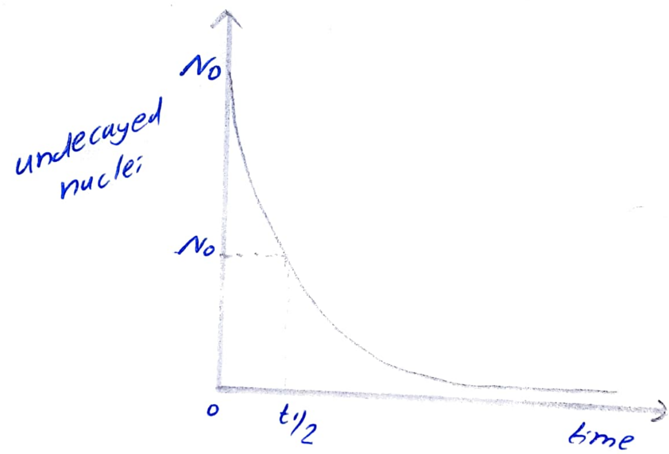
where λ is the decay constant. Analytically solving it shows that decay is exponential

THEORY

Radioactive decay is a spontaneous and process where in unstable nuclei disintegrate over time, following are exponential decay law. The number of undecayed nuclei at any time t is given by solving the above differential equation, $N(t) = N_0 e^{-\lambda t}$ where N_0 is the initial number of nuclei and λ is the decay constant. The decay constant physically signifies the probability that are particular atom undergoes radioactive decay in a infinitesimal time step 'dt'

In this experiment, probabilistic decay is modeled using Monte-Carlo method. At each time step, each nucleus is assumed to have a fixed probability of decaying. By simulating a large number of such decay ~~events~~ events are successive intervals, the aggregate behaviour statistically approximates the exponential decay curve

Graph of analytical solution (expected)



Algorithm and flowchart

Step 1: Import required libraries

Step 2: Set initial parameters

$N_0 = 1000$, $\lambda = 0.06966$ (N-13), steps = 100

Step 3: Calculate theoretical half-life

$$t_{1/2} = \frac{\ln(2)}{\lambda}$$

Step 4: Initialize array & variable decay_mc to store remaining particles after each step

t_half_mc = none

Step 5: Monte-Carlo simulation of decay for each time 't' from 0 to step 1

i) Generate N random numbers from 0 to 1

ii) Count how many are less than λ. There present decayed ~~from~~ particles

iii) Subtract the number of decayed from N

iv) Store the current N in decay-mc[t]

v) If half_mc hasn't been set and N has dropped below half of N0, set t_half_mc = t

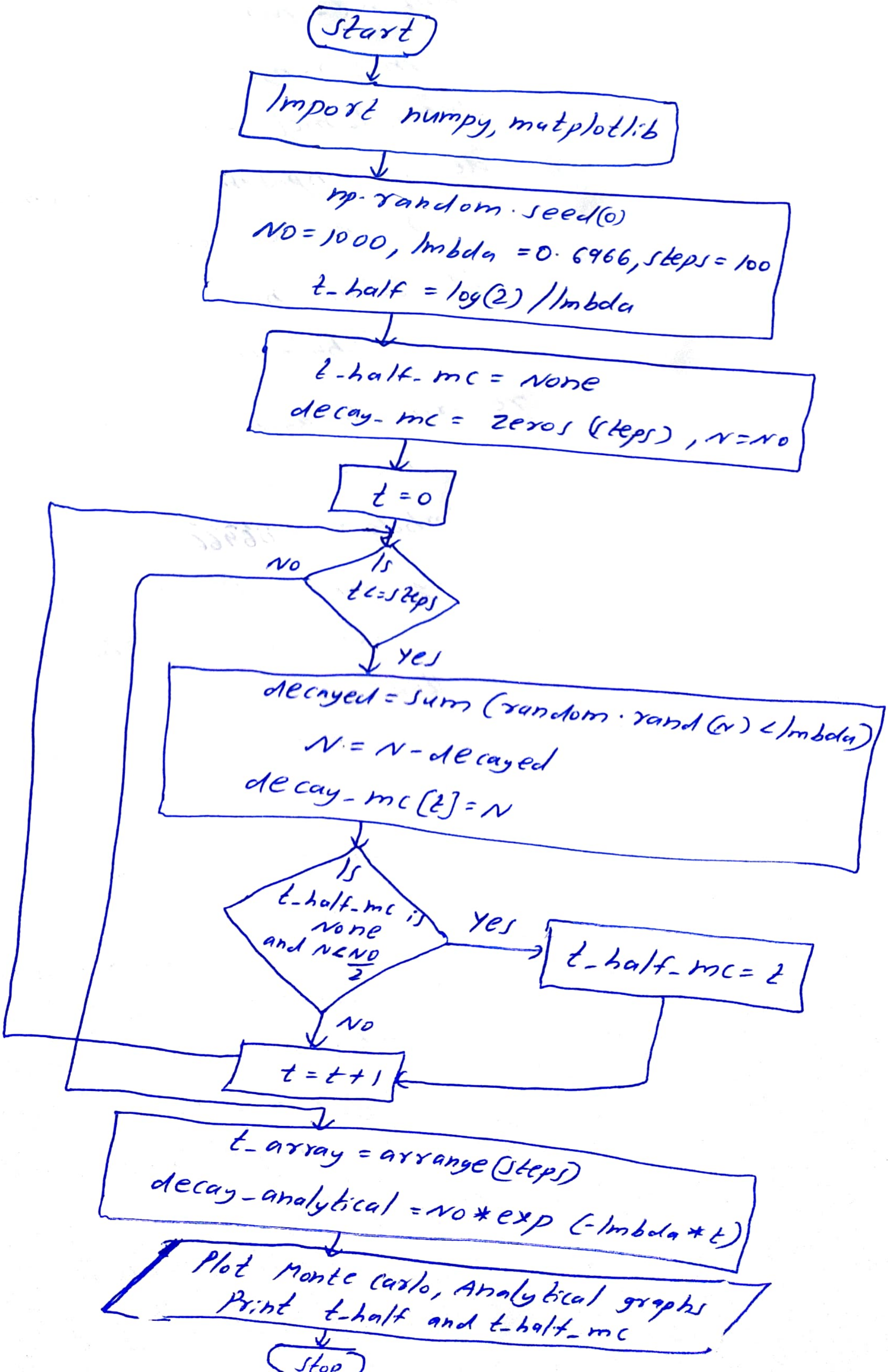
Step 6 : Analytical solution

$$N(t) = N_0 e^{-\lambda t}$$

Step 7: Plot the results

Step 8: Compare theoretically obtained halfalife & Monte-carlo half-life

flowchart



PROCEDURE - CODE

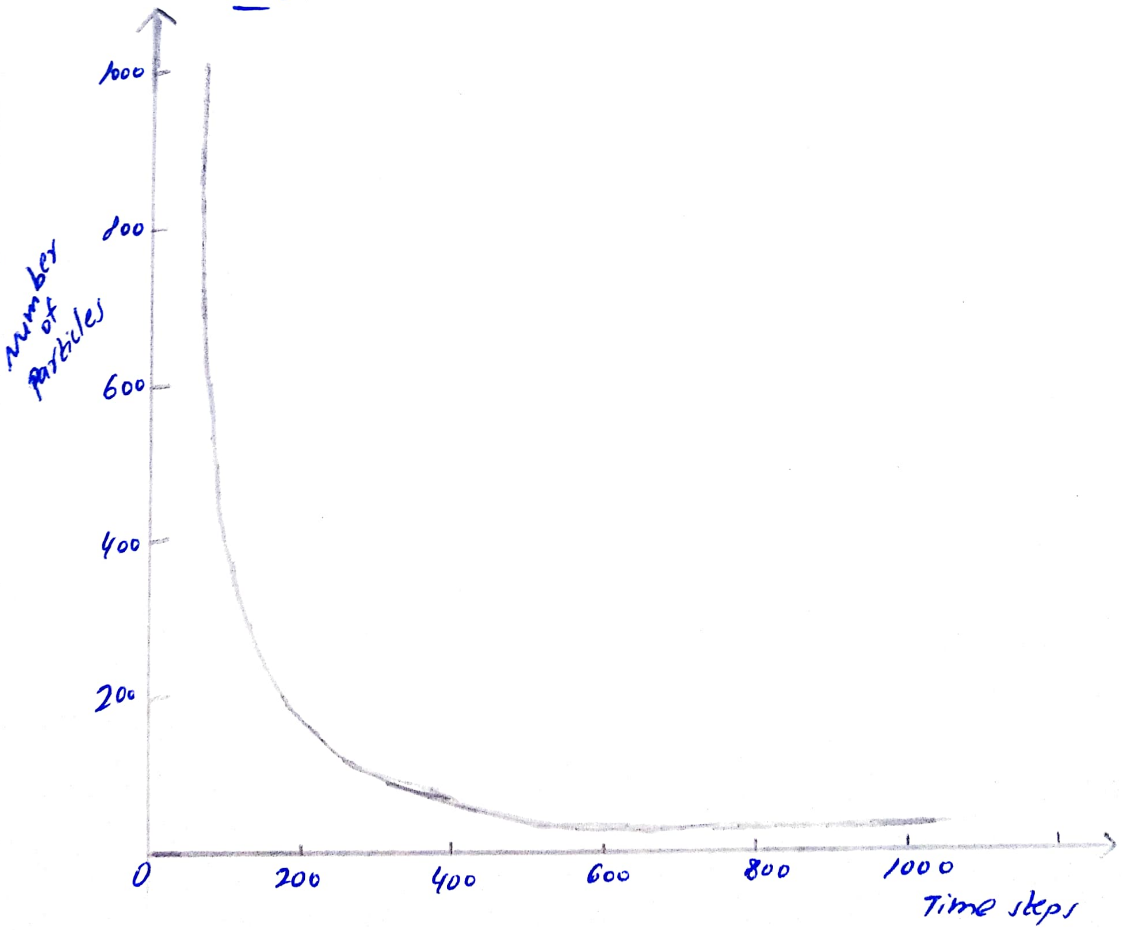
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)
def monte_carlo_decay_simulation(~~no~~ no, lmbda, steps):
    decay_mc = np.zeros(steps)
        n = no
      t_half_mc = None
    for t in range(steps):
      decayed_mc[t] = n
      decayd = np.sum(np.random.rand(n) < lmbda)
        n = n - decayed
      if t_half_mc is none and n < no/2:
          t_half_mc = t
    return decay_mc, t_half_mc
        no = 1000
      lmbda = 0.01023
    #decay constant of N-13 = 0.06966
    #decay constant of Ga-68 =                0.01023
      steps = 1000
decay_mc, t_half_mc = monte_carlo_decay_simulation
                          (no, lmbda, steps)
     t_half = np.log(2)/lmbda
   time_array = np.arrange(steps)
   decay_analytical = no * np.exp(-lmbda * time_array)
   plt.figure(figsize=(10,6))
   plt.plot(time_array, decay_mc, label='Monte
          Carlo simulation', color='blue',
          marker='o', markersize=3)
   plt.plot(time_array, decay_analytical,
          label='Analytical solution', color='red',
          linestyle='--')
```

```
plt.xlabel('Time steps')
plt.ylabel('Number of particles')
plt.title('Monte Carlo Simulation of
            Radioactive Decay')
plt.xlim(0,300)
plt.grid(True)
plt.legend()
plt.show()

print("Simulated Half-life :", t_half_mc)
print("Theoretical Half-life :", t_half)
```

RESULT



Simulated Half-life : 65

Theoretical Half-life : 67.7563