PHY 607 - Project 2 Proposal: Tritium Decay

Caroline Capuano and Jasmine Watt

October 2024

1 Introduction

1.1 Description of the simulation

Using the Monte Carlo simulation, we can model tritium decay for carbon dating purposes. Tritium is a radioactive isotope of hydrogen, or known as ^3_1H. The expected half life of Tritium is 12.33 years and as Tritium decays. The decay of Tritium can be represented as the following:

$$^3_1He \rightarrow ^3_2 He + e^- + \overline{v}_e \tag{1}$$

Using the decay and half life of Tritium, we can find the decay probability of Tritium. The key factor of the decay is the half life of Tritium. Manipulating the decay time, would directly impact the decay observed. This can validate our findings after using the Monte Carlo Simulation. The distribution of the decay probability will also be specific to the decay of Tritium. Finding this decay probability will tell us about the properties of Tritium.

Using Schrödinger's Equation, we can also determine the energy of Tritium. The wave function of tritium can be written as a hydrogen like atom:

$$\phi_o = \frac{1}{\sqrt{\pi}}(\frac{Z}{a_o})^{\frac{3}{2}}e^(\frac{-Zr}{a_o}) \tag{2}$$

This wave function satisfies the Schrödinger's Equation. This will also show us information about the properties of Tritium. Using Schrödinger's Equation also satisfies the requirement of using a second order ODE for the assignment. The second order ODE Schrödinger's Equation is as following:

$$\frac{-\hbar}{2m}\nabla^2\psi + V(x)\psi = E\psi \tag{3}$$

1.2 High-level psuedocode

First, we want to use the Monte Carlo simulation method. Using random numbers, we can portray the decay of Tritium. Firstly, we want to set key properties of the decay of Tritium, like the half life and decay constant.

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

# constants of Tritium
half_life = 12.32 # half-life of Tritium in years
decay_constant = np.log(2) / half_life # final amount of tritium after
    12.32 years
```

Then, we would set initial parameters for the simulation like the initial number of atoms, duration of the simulation, the amount of steps taken and the time step of the simulation (dt)

```

# initial parameters
initial_atoms = 500
steps = 100
duration = 100 # years
dt = duration / steps # time step size
```

Then, we will set up empty arrays for the total time of decay and the amount of atoms leftover after the decay. Then, we will set up the Monte carlo simulation where the decay equation will be passed through random numbers and the results will be stored in the predetermined array.

```


# arrays for results
time = np.linspace(0, time_duration, time_steps)
final_atoms = np.zeros(time_steps)

# monte carlo Simulation
for t in range(time_steps):
    decay_happens = 0

    for i in range(initial_atoms):
        if np.random.random() < (1 - np.exp(-decay_constant * dt)):
            i = i + 1

    final_atoms[t] = initial_atoms - decay_happens
    initial_atoms = final_atoms[t]

# plot results
```

The next step is to solve the Schrödinger's Equation using the predetermined wave function for Tritium. Same as before, first we will define constants for the Schrödinger's Equation.

```
hbar = 1.0545718e-34 # planck's constant in Js
m_e = 9.10938356e-31 # mass of an electron in kg
e = 1.602176634e-19 #  charge in C
epsilon_0 = 8.854187817e-12
Z = 1 # Atomic number for hydrogen
```

We could then find the probability density using the wavefunction of Tritium. Using numpy.linalg.eigh(), we can find the eigenvalues and eigenvectors to solve for the probability density. Using the time independent Schrödinger's Equation, we can see how the probability density also evolves over time.

To satisfy the classes requirement, we could develop a class for the Hamiltonian for the Schrödinger's Equation, and a separate class to solve the Equation. Similiarly, we could determine a class for the Tritium Decay and for the probability, like the following:

```
class SchrodingerSolver:
    def __init__(self, potential, x_min, x_max, num_points):
            #begin Schrodinger's Equation here with the potential and
                hamiltonian.

    def _create_hamiltonian(self):
        #return hamiltonian here

    def solve(self):
        # solve schrodinger's equation here
-----
class TritiumDecay:
    def __init__(self, half_life=12.32):
        self.half_life = half_life
            self.decay_constant = np.log(2) / self.half_life

    def decay_probability(self, time):
        probability = 1 - np.exp(-self.decay_constant * time)
        return probability
```

1.3 Work distribution

1.3.1 Jasmine

1. Use the Monte Carlo simulation method to plot the decay of Tritium
2. Determine potential needed for Schrödinger's Equation
3. Work on the introduction and results section of the report
4. Assess the errors introduced by finite precision, numerical integration, and statistical sampling for report

1.3.2 Caroline

1. Test constraints of the decay of Tritium to validate results
2. Develop classes for the Schrödinger's Equation
3. Use numpy.linalg.eigh() to find the eigenvalues and function of the equation
4. Plot the probability density from Schrödinger's Equation
5. Discuss the results the of the simulation run for at least three levels of detail.

1.4 Key Points of Project

Notes:

Define the Problem: Identify the system or process you want to model, including the variables involved.

Pseudocode

Determine Input Variables: Establish which input variables are uncertain and define their probability distributions (e.g., normal, uniform, triangular).

Generate Random Samples: Use random number generators to create samples of the input variables based on their defined distributions.

Run Simulations: Input the random samples into the model and compute the output for each set of samples. This is typically done thousands or millions of times to build a comprehensive data set.

Analyze Results: Compile the output data to analyze the distribution of possible outcomes. This might involve calculating mean values, variances, and confidence intervals.

Make Decisions: Use the insights gained from the simulation to inform decision-making, risk management, or strategic planning.