PHY 607 - Project 2: Using Schrödinger's Equation to Find the Wave Function for Tritium

Caroline Capuano and Jasmine Watt

October 2024

1 Introduction

Tritium is a radioactive isotope of hydrogen, or known as ${}^3_1H$. Using Schrödinger's Equation, we can determine the wave function of Tritium. The wave function of tritium as the following, where Z is the number of protons:

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

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

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

2 Methods

To solve for the second differential of the wave function, we used the 4th Order Runga-Kutta method. The 4th Order Runga-Kutta method was chosen to differentiate the wave function because it proved to be the more accurate method in Project 1. Firstly, we defined the constants needed to solve Schrödinger's Equation.

```
hbar = 1.0 # planck's constant (Js)
m = 3.0    # mass of tritium (kg)
```

Then, an infinite square well was used to determine the potential energy inside and outside the well. When the outside potential was set to be infinite, an error was produced because np.inf was not allowed to be a variable. Instead, a potential of 10,000,000 was used to replicate the infinite square well.

```
class Potential:
    def __init__(self, width=1):
            self.width = width

        def value(self, x):
            if 0 < x < self.width:
                return 0
            else:
                return 10000000 # Infinite potential outside the well
```

Then, to begin solving Schrödinger's Equation, a new class was established called WaveFunctionSolver. This class called the previous potential that was determined in the previous class, and began applying Schrödinger's Equation.

```
    class WaveFunctionSolver:
        def __init__(self, potential):
            self.potential = potential

        def derivatives(self, x, y, E):
            psi1 = y[0]
            psi2 = y[1]
            V = self.potential.value(x)
            dpsi1_dx = psi2
            dpsi2_dx = (2 * m / hbar**2) * (V - E) * psi1
            return np.array([dpsi1_dx, dpsi2_dx])
```

To solve the second derivative portion of Schrödinger's Equation, we used the 4th Order Runga-Kutta method to approximate the second derivative.

```
    def runge_kutta(self, E, y0, x0, x_end, dx):
        num_steps = int((x_end - x0) / dx) + 1
        x_values = np.linspace(x0, x_end, num_steps)
        y_values = np.zeros((num_steps, len(y0)))
        y_values[0] = y0

        for i in range(num_steps - 1):
            x = x_values[i]
            y = y_values[i]

            k1 = dx * self.derivatives(x, y, E)
            k2 = dx * self.derivatives(x + dx / 2, y + k1 / 2, E)
            k3 = self.derivatives(x + dx / 2, y + k2 / 2, E) * dx
            k4 = self.derivatives(x + dx, y + k3, E) * dx

            y_values[i + 1] = y + (k1 + 2 * k2 + 2 * k3 + k4) / 6

        return x_values, y_values
```

A function "eigh" from SciPy was also used to find the eigenvalues and eigenfunctions of this wave form.

```
def solve(self): # solve eigenvalues using SciPy script eigh
    energies, wavefunctions = eigh(self.hamiltonian)
    return energies, wavefunctions
```

Lastly, the Monte-Carlo simulation method of using random x-values is used in this following code. To normalize the function, we also had to apply the trapezoidal rule to estimate the integral because the SciPy integrate function would not work.

```
for _ in range(self.num_samples):
        # generate a random x value within the specified range
        random_x = np.random.uniform(self.x0, self.x_end)
        x_values, solution = self.solver.runge_kutta(self.E,
            np.array([1, 0]), self.x0, self.x_end, self.dx)

        # find the corresponding psi value for the random x
        psi = solution[:, 0]
        norm = np.sqrt(np.trapz(np.abs(psi)**2, x_values))
        psi /= norm
```

3 Results

Figure 1 shows the simulated probability density of Tritium after using Monte Carlo simulation for randomness and the 4th order Runga-Kutta to estimate the second derivative of the wave function.

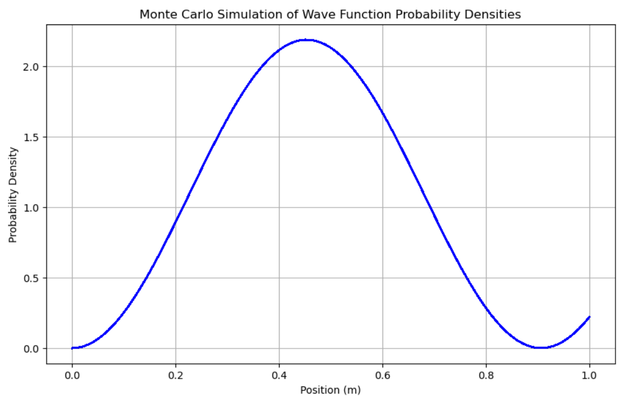

Figure 1: Simulation of the Probability Density of Tritium after using Monte Carlo simulation for randomness and the 4th order Runga-Kutta

The width of the infinite square well was from 0 to 1. Some interesting things can be seen in the probability density. As to be expected, the particle will most likely be found around the center of the well. This makes sense because it is more energetically favorable to be inside the well than outside because of the determined parameters of the well. However, at the end of the well you can see a small peak in the probability density.

4 Validation

Using an infinite square well for the potential of the Wave function allows us to manipulate the probability density for the wave function. The infinite square well is a potential energy well that breaks up the space between two potentials: the potential is zero inside the well and infinity outside the well. Because the potential is zero inside the cell, the particle is more likely to be found inside the well because the particle can exist freely without an energy barrier. Outside the well, the potential is infinite ensuring that the particle would not be outside the well. The probability density of Tritium for an infinite square well can be seen in Figure 1.

Changing the potential at the boundaries of the well should change the potential density of Tritium. By decreasing the potential barrier to 1, the potential density distribution should change inside the well and at the barriers.

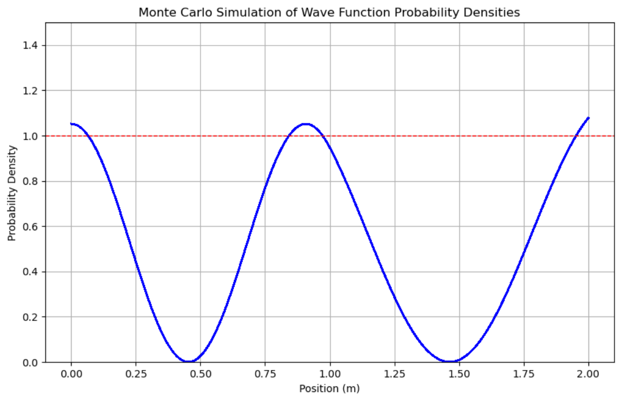

Figure 2: Estimated Probability Density with Potential Barrier of V = 1

The probability distribution does certainly change inside the well and at the barriers, as can be seen in Figure 2. Tritium is not least probable at the center of the well, and most probable at the potential barrier. Interestingly, the probability even increases inside the potential well. The red line was added to observe the peak probability density increasing.

5 Accuracy

To observe the accuracy of our simulation, we also solved the Schrödinger's Equation with the actual wave function from Tritium.

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

Using the 4th order Runga-Kutta again to estimate the second derivative of the wave function, we were able to plot our estimated probability density and the actual probability density of Tritium in Figure 3.

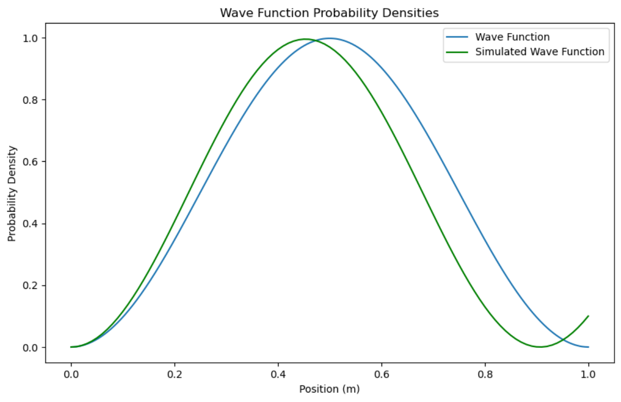

Figure 3: Comparison between the Simulated Probability Density and the Actual Probability Density of Tritium

As you can see, the simulation tends to get less accurate the longer the simulation goes on. This could be due to the fact that Runga-Kutta was used to estimate the second derivative. The 4th order Runga-Kutta method has an error on the scale of the $\mathcal{O}(h^4)$. This means that the estimation gets less accurate the towards the end of the approximation.

Figure 4 was created by finding the difference between the simulated probability density and the actual probability density from Tritium's Wave function.

To find this error, the following code was implemented:

```
difference = normalized_wavefunctions[:, i]**2 -
    np.abs(psi_interp)**2
x_differences = np.diff(quantum_system.x)
plt.plot(quantum_system.x[:-1], np.abs(difference[:-1]) color='red')
```

A couple of errors I ran into to find the difference between the two plot was making sure both wave function arrays were the same length, and calculating the difference between the x values. After fixing those two errors, Figure 4 was produced.

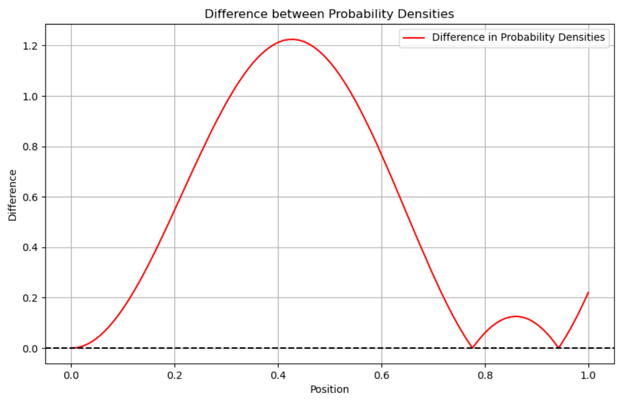

Figure 4: Error of the Simulated Probability Density

The reason I wanted to explain this process was because the error plot was not what I was expecting. The difference between the two plots seems to increase as the x values increase in Figure 3. However, Figure 4 shows that the simulation had the biggest error in the center of the well, decreases, then has a little spike. The little spike around 0.8 m was to be expected because that was observed in the simulated probability density and not the actual probability density. However, the error should have increased throughout the well instead of increasing then decreasing. This could have been an error in re-arranging the arrays so that they would have the same length, but I did not have time to find the error.

6 Conclusion

Using Schrödinger's Equation, we simulated the wave function of Tritium. Using Runga-Kutta, the trapezoidal rule, and SciPy we were able to model the wave function of Tritium. We then compared our simulation to the actual wave function of tritium which is the following (where Z is the number of protons):

$$\phi_o = \frac{1}{\sqrt{\pi}}(\frac{Z}{a_o})^{\frac{3}{2}}e^{(}\frac{-Zr}{a_o}) \tag{4}$$

We were able to simulate a pretty accurate probability density plot for Tritium, then also tested our findings by changing the potential well the particle was in from an infinite potential well to a finite potential well of very low potential.