Hydrogen_Atom_Ground_State

January 4, 2024

#090200705 Beria Ayşenur Can

a) The probability density for finding an electron in a hydrogen atom in the ground state, which involves plotting a graph of the probability density as a function of radial distance (r/a0).

b) Determining the most probable location of an electron in the ground state, which would likely involve finding the maximum of the probability density function.

c) Finding the expected value of the position for the ground state wave function, and indicating this on the graph.

[51]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.constants import hbar, m_e ,pi
from scipy.optimize import minimize_scalar
from scipy.integrate import quad
```

#a)

$$\psi_{100}(r) = \frac{1}{\sqrt{\pi \cdot a_0^3}} \cdot e^{-\frac{r}{a_0}} \tag{1}$$

$$dP = |\psi|^2 \cdot 4\pi \cdot r^2 \cdot dr \tag{2}$$

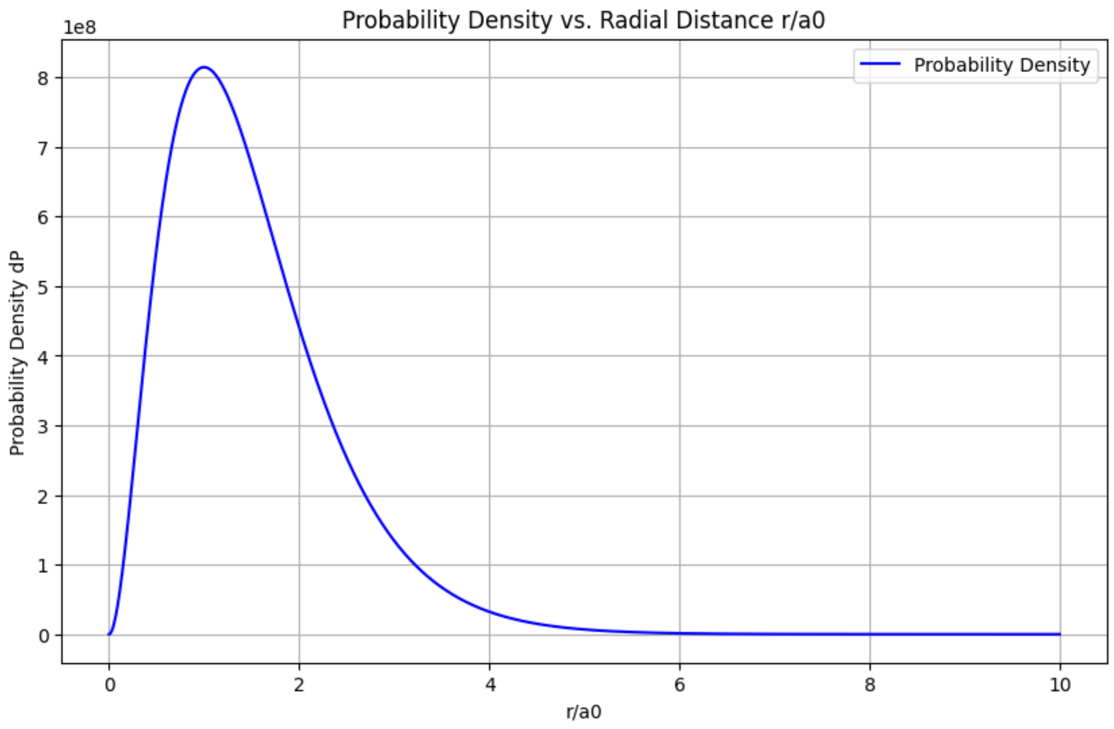
$$dP/dr = |\frac{1}{\sqrt{\pi \cdot a_0^3}} \cdot e^{-\frac{r}{a_0}}|^2 \cdot 4\pi \cdot r^2 \tag{3}$$

$$dP/dr = |\frac{1}{\sqrt{\pi \cdot a_0^3}} \cdot e^{-\frac{r}{a_0}}|^2 \cdot 4\pi \cdot r^2 \tag{4}$$

[48]:
```python
a0 = 0.0529e-9 # bohr radius


# wave function
def psi_100(r):
    return (1 / np.sqrt(np.pi * a0**3)) * np.exp(-r / a0)
```

```
# dP
def probability_density(r):
    return np.abs(psi_100(r))**2 * r**2

# from 0 to 10 times the Bohr radius
r = np.linspace(0, 10*a0, 1000)

dP = probability_density(r)
```

[49]:
```
plt.figure(figsize=(10, 6))
plt.plot(r/a0, dP, label='Probability Density', color='blue')
plt.xlabel('r/a0')
plt.ylabel('Probability Density dP')
plt.title('Probability Density vs. Radial Distance r/a0')
plt.legend()
plt.grid(True)
plt.show()
```



#b)

$$\frac{d}{dr}\left(\frac{dP}{dr}\right) = 0 \tag{5}$$

$$\frac{d}{dr}\left(\frac{dP}{dr}\right) = \frac{4}{a_0^3} \cdot (2r \cdot e^{-\frac{2r}{a_0}} \cdot (1 - \frac{r}{a_0})) \tag{6}$$

[43]:
```
def probability_density_derivative(r, a0):
    # Derivative of probability density function with respect to r
    return (4/a0**3) * (2 * r * np.exp(-2*r/a0) * (1 - r/a0))
```

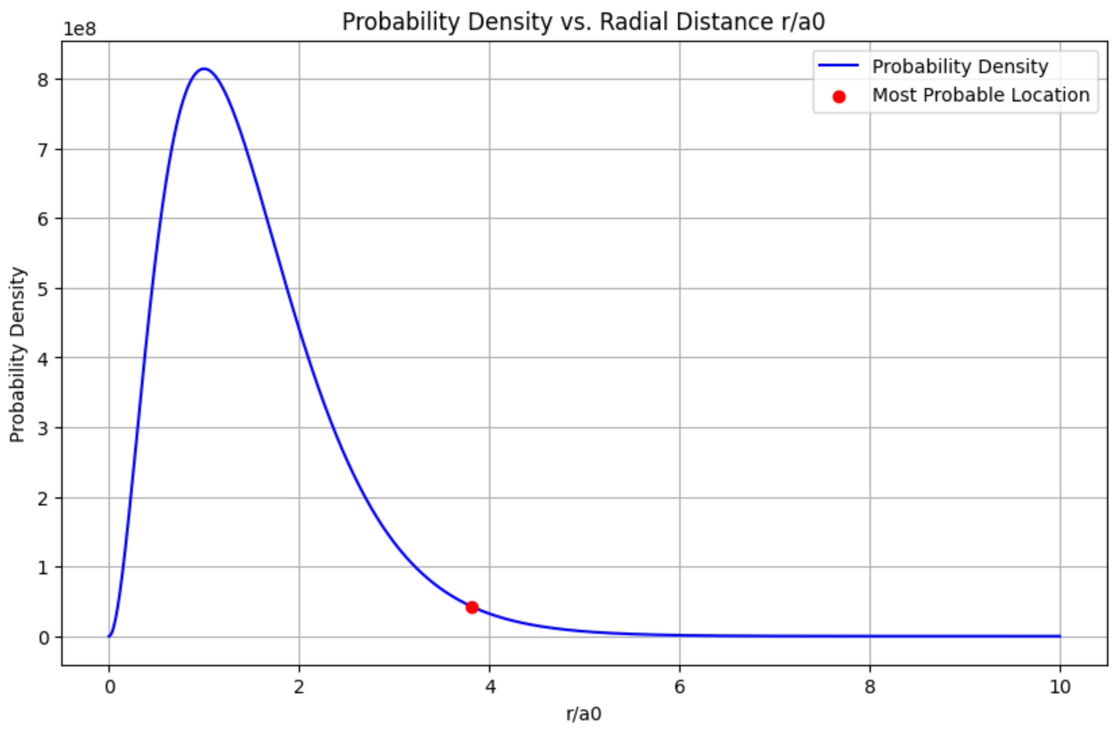
[44]:
```
def root_function(r):
    # We want to find where this derivative is zero
    return probability_density_derivative(r, a0)
```

minimize_scalar fonksiyonuna negatif değer yerine türevin karesini kullanarak sıfıra en yakın noktayı bulduk. Bu, türevin işaretinden bağımsız olarak doğru kökü bulmamızı sağlar, çünkü bir fonksiyonun karesi her zaman pozitiftir ve sıfıra en yakın nokta fonksiyonun kökü olacaktır

[45]:
```
result = minimize_scalar(lambda r: root_function(r)**2, bounds=(0, 10*a0),␣
 ↪method='bounded')
r_max = result.x
r_max_normalized = r_max / a0  # Normalizing the most probable location by Bohr␣
 ↪radius

r_max, r_max_normalized
```

[45]:
```
(2.0206001995130562e-10, 3.8196601125010514)
```

[47]:
```
# Normalized radial values for plotting
r_values = np.linspace(0, 10*a0, 1000)
dP_values = probability_density(r_values)

# Plotting the probability density and marking the most probable location
plt.figure(figsize=(10, 6))
plt.plot(r_values / a0, dP_values, label='Probability Density', color='blue')
plt.scatter(r_max / a0, probability_density(r_max), color='red', zorder=5,␣
 ↪label='Most Probable Location')
plt.xlabel('r/a0')
plt.ylabel('Probability Density')
plt.title('Probability Density vs. Radial Distance r/a0')
plt.legend()
plt.grid(True)
plt.show()
```

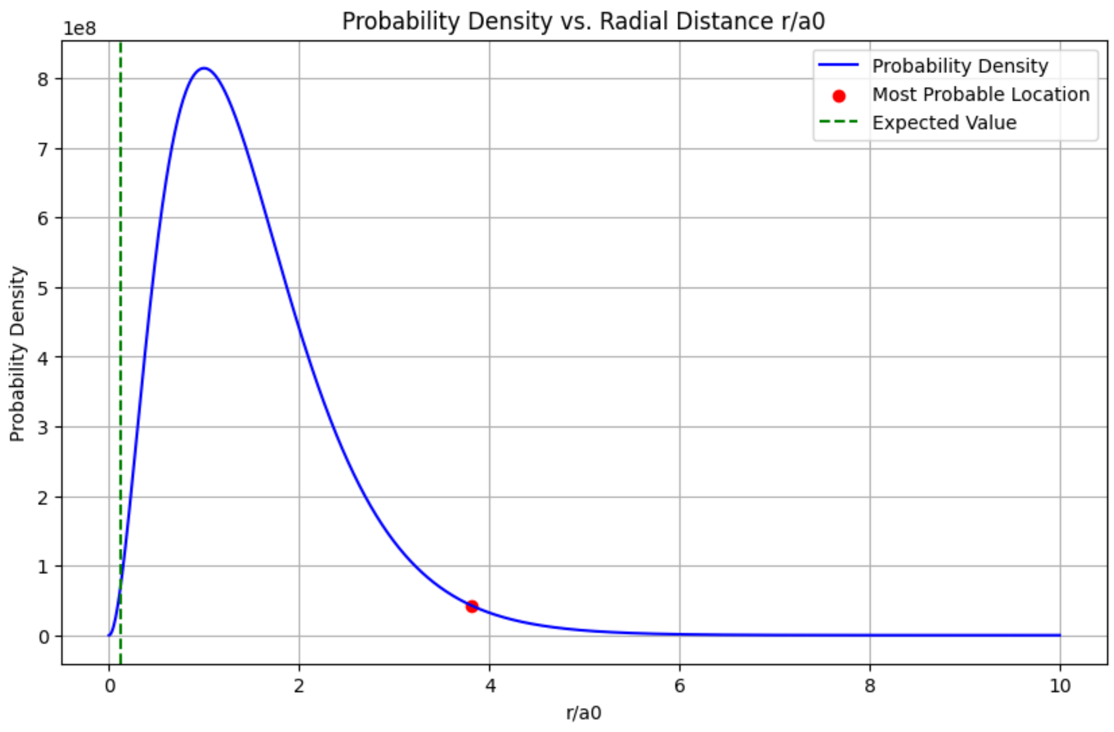


#c)

$$\langle r \rangle = \int_0^\infty r \cdot dP \, dr \tag{7}$$

[52]:
```
# Define the integrand for the expected value of r
def integrand_for_expected_r(r):
    return r * probability_density(r)


# Compute the expected value of r (mean distance) using numerical integration
# We'll integrate from 0 to a point where the wave function is effectively zero
upper_limit = 20 * a0  # Integration upper limit
expected_r, _ = quad(integrand_for_expected_r, 0, upper_limit)

# Normalizing the expected value by the Bohr radius
expected_r_normalized = expected_r / a0

# Add the expected value to the plot
plt.figure(figsize=(10, 6))
plt.plot(r_values / a0, dP_values, label='Probability Density', color='blue')
plt.scatter(r_max / a0, probability_density(r_max), color='red', label='Most␣
↪Probable Location')
```

```
plt.axvline(x=expected_r_normalized, color='green', linestyle='--',␣
 ↪label='Expected Value')
plt.xlabel('r/a0')
plt.ylabel('Probability Density')
plt.title('Probability Density vs. Radial Distance r/a0')
plt.legend()
plt.grid(True)
plt.show()

expected_r_normalized, expected_r
```



```
[52]: (0.11936620731891535, 6.314472367170622e-12)
```