Exploration of the Schrödinger Equation for a Particle in a Finite Potential Box

Lynn Aung

December 9, 2024

I Introduction

The Schrödinger equation is one of the most significant equations of quantum mechanics, giving us a mathematical framework for elucidating the quantum state of physical systems. This equation allows physicists to predict the behavior of particles at atomic and subatomic scales, revolutionizing our understanding of the universe.

This brief project will explore solutions to the three-dimensional time-independent Schrödinger equation for a particle confined in a finite potential box. The objective is to visualize quantized energy levels through the use of heatmaps and 3-dimensional models. By focusing on the hydrogen atom, this work aims to enhance comprehension of the interplay between quantum mechanics and physical phenomena.

Unlike the infinite potential box, where the potential walls are infinitely high, a finite potential box allows for a more realistic model where the particle can penetrate and even tunnel through the potential barriers, albeit with diminishing probability. This difference shows significant changes in the energy levels and wave functions compared to the infinite case.

We use the method of separation of variables to solve the Schrödinger equation. First, we assume that the wavefunction can be shown as a product of three independent functions, each dependent on one spatial coordinate, x, y, or z. This results in three coupled differential equations, which are solved under the boundary conditions that the wave function approaches zero outside the potential walls but does not necessarily vanish sharply at the walls.

The quantized energy levels for a finite potential box are influenced by the depth and width of the potential well, which leads to non-uniform spacing between the energy state levels. This becomes more notable when we go to higher energies. Although the ground state energy is higher compared to the infinite potential box due to the finite height of the potential walls.

To visualize the wavefunction probability densities, I look at each selected energy state. For instance, the ground state, where $(n_x = 1, n_y = 0, n_z = 0)$, exhibits the highest probability density near the center of the box, gradually decreasing towards the edges. Higher energy states, such as $(n_x = 2, n_y = 1, n_z = 1)$, show more complex structures with nodes, indicating regions where the probability density is zero. This visualization is intuitive; if you were to guess the location of the particle inside the box, it would most likely be near the center, reflecting the higher probability density in that region.

To provide a graphical representation, I generated heat maps for the probability densities of these states, illustrating where the particle is most likely to be found within the box. These visualizations, combined with the numerical solutions, give insights into the behavior of particles in finite potential wells. This study highlights the quantized nature of energy in confined systems and the importance of potential barrier characteristics in determining energy states, demonstrating the fundamental principles of quantum mechanics through the analysis and visualization of solutions to the Schrödinger equation.

II History of the Schrödinger Equation

The Schrödinger equation was created by Austrian physicist Erwin Schrödinger in 1926 and it was developed during the early periods of quantum mechanics. Schrödinger was trying to understand the wave-particle duality of matter, which was first suggested by Louis de Broglie in 1924. In simple terms, De Broglie said that particles like electrons have waves-like properties, with a wavelength that's inversely proportional to their momentum.

Building on this idea, Schrödinger came up with a wave equation to describe how quantum particles behave. The time-independent form of the equation is:

$$-\frac{\hbar^2}{2m}\nabla^2\psi + V\psi = E\psi,$$

where ψ is the wave function, V is the potential energy, E is the energy eigenvalue, and $\hbar$ is the reduced Planck constant, governs the spatial distribution of the particle's probability density. This wave function's square magnitude, $|\psi|^2$, shows the probability of finding the particle at a specific location.

The equation changed physics by introducing quantization. It explained phenomena such as electron orbitals in atoms, the emission spectra of gases, and tunneling effects, laying the foundation for quantum mechanics.

Significance of This Exploration Solving the Schrödinger equation in three dimensions for a finite potential box—which is often called a quantum well or particle-in-a-box problem—has a direct impact on understanding quantized energy states. These quantized states are the foundation of electron behavior in atoms, molecules, and condensed matter systems. By focusing on the hydrogen atom, this study not only explores the basics of quantum mechanics but also connects the dots between abstract equations and their physical manifestations.

The visualizations, like heatmaps of probability distributions, are great educational tools that help people understand quantum behavior better. This project also shows how differential equations and numerical solutions are used in modern physics research.

III Why We Are Using Hydrogen for This Study

The hydrogen atom is the simplest atom in the periodic table. This means that the hydrogen atom consists of only one proton in its nucleus and has only one orbiting electron. This makes it simple for us to explore quantum mechanical principles with ease and clarity. There are several reasons why hydrogen is the best atom to run this project on:

1. Simplicity of the System Hydrogen's single electron configuration makes it easier for me to calculate with hydrogen because it's simple. With only one electron, the Schrödinger equation is much simpler, reducing the computational complexity that typically challenges multi-electron systems. This means we can get analytical solutions without using complex numerical techniques. We can write out the wave function for a hydrogen atom. This lets us calculate energy levels, probability distributions, and spatial configurations. Atoms like helium or lithium require more complex calculations. But hydrogen makes it easier to understand quantum mechanics.

Additionally, with fewer variables to process, my calculations become more straightforward, and the processing load on my computer decreases dramatically. This means lighter memory usage, faster processing, and more efficient algorithm execution. Compared to multi-electron atoms that require complex numerical approximations, hydrogen allows for direct analytical solutions that are not only mathematically simple but also computationally lightweight. As a novice programmer, my visualizations are not the most optimized. However, the current approach is already at $O(n^3)$, which is suboptimal. If I did other atoms, it would significantly strain my CPU.

2. Exact Analytical Solutions The time-independent Schrödinger equation for hydrogen yields precise analytical solutions, which could serve as a benchmark for numerical methods used in solving quantum mechanical problems. Multi-electron systems unlike the hydrogen atom usually requires approximation techniques like Hartree-Fock or density functional theory, which can hallucinate quantum mechanical principles.

3. Quantum Number Representation The Schrödinger equation for the hydrogen atom reveals a set of quantum numbers—n, l, and m—that uniquely describe the electron's behavior (states) and properties in this system. These quantum numbers shows the electron's state within the atom.

- The **principal quantum number** n determines the energy level of the electron and the overall size of its orbit, with higher values of n corresponding to electrons farther from the nucleus.
- The **orbital angular momentum quantum number** l defines the shape of the electron's orbital, with values ranging from 0 to $n-1$. Each l-value corresponds to a different orbital type (e.g., $l = 0$ for spherical s-orbitals, $l = 1$ for dumbbell-shaped p-orbitals, etc.).
- The **magnetic quantum number** m specifies the orientation of the orbital in space relative to an external magnetic field, with possible values ranging from $-l$ to $+l$.

5. Relevance to Real-World Applications Despite its atomic simplicity, hydrogen serves as a foundational model in multiple scientific domains, including spectroscopy, astrophysics, and plasma physics. The principles derived from hydrogen's quantum mechanical behavior can be systematically extended to understand more complex atomic and molecular interactions.

6. Consistency Across Potential Models In this project, the hydrogen atom's electron is visualized within a finite potential box, maintaining a consistent framework for showing quantized states and their changes in constrained systems. This approach allows controlled methodology for visualizing and interpreting quantum mechanical principles.

7. Well-Documentation & Interpretation The probability distributions and energy levels of hydrogen are well-documented and easy to understand. These visualizations help researchers and students understand quantum mechanical systems.

Thus, studying the hydrogen atom shows its value as a quantum mechanical model. It connects theoretical understanding with practical scientific insight.

IV Methodology: Quantum Mechanics of a Particle

IV.I Schrödinger's Equation and the Laplacian Operator As we have told in the beginning, we have the Schrödinger's Equation.

$$-\frac{\hbar^2}{2m}\nabla^2\psi + V\psi = E\psi,$$

Here, the term $-\frac{\hbar^2}{2m}\nabla^2\psi$ represents the kinetic energy of the particle and the term $V\psi$ represents the potential energy of the particle, which add up to $E\psi$ which is the total energy of the particle.

Next, if we look at the original equation, we see the ∇^2. This is what we call the Laplace operator in Schrödinger's equation. The Laplace operator, also known as the Laplacian, is a second-order differential operator that describes the flux density of the gradient flow of a function. In Cartesian coordinates, the Laplacian is expressed as the sum of second partial derivatives with respect to each independent variable:

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$$

The Laplacian is used in Schrödinger's equation for several reasons:

- **Kinetic Energy:** In quantum mechanics, the Laplacian represents the kinetic energy operator. The Schrödinger equation describes the total energy of a quantum system, which includes both kinetic and potential energy.
- **Wave-like Nature:** The Laplacian appears in wave equations, and the Schrödinger equation describes the wave-like behavior of quantum particles.
- **Probability Distribution:** The Laplacian helps describe how the probability distribution of a particle's position changes over time in quantum mechanics.
- **Mathematical Framework:** The Laplacian provides a convenient mathematical framework for describing the behavior of quantum systems in three-dimensional space.

IV.II Boundary Conditions and Potential Energy Next, we substitute the sum of second partial derivatives into Schrödinger's equation and distribute:

$$-\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}\right)\psi+V\psi=E\psi,$$

$$-\frac{\hbar^2}{2m}\left(\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}+\frac{\partial^2\psi}{\partial z^2}\right)+V\psi=E\psi,$$

The wave function is required to be continuous, so $\psi = 0$ on the sides of this cube, that is, the $x = 0$, $x = L_x$, $y = 0$, $y = L_y$, $z = 0$, and $z = L_z$ planes.

Inside the cube, then, the initial boundary value problem to solve is:

$$E\psi=-\frac{\hbar^2}{2m}\left(\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}+\frac{\partial^2\psi}{\partial z^2}\right)$$

You might notice that the term for the potential energy is missing in this equation and that is because there is no potential energy within the box. The potential energy function for a particle in a box is:

$$V(x)=0\ for\ 0\ <x<L$$

$$V(x)=\infty\ for\ x\leq 0\ or\ x\geq L$$

where L is the length of the box. Inside the box $0 < x < L$, the particle moves freely without any forces acting upon it. This means that the potential energy term V(x) in the Schrödinger equation becomes zero.

This simplification allows us to solve the equation more easily, focusing on the kinetic energy term and the total energy of the system. The infinite potential at the boundaries ensures that the wavefunction goes to zero at $x = 0$ and $x = L$, providing the necessary boundary conditions for quantization of energy levels.

Inside the cube, then, the initial boundary value problem to solve is:

$$E\psi=-\frac{\hbar^2}{2m}\left(\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}+\frac{\partial^2\psi}{\partial z^2}\right),\ 0<x,\ y,\ z<L$$

IV.III Separation of Variables Since Schrödinger's equation and its associated boundary conditions are linear and homogeneous, we can use the method of separation of variables:

$$\psi(x,y,z)=X(x)Y(y)Z(z)$$

We can use this product solution of the wavefunction and substitute it in Schrödinger's equation, giving us:

$$E[X(x)Y(y)Z(z)]=-\frac{\hbar^2}{2m}\left(\frac{\partial^2(X(x)Y(y)Z(z))}{\partial x^2}+\frac{\partial^2(X(x)Y(y)Z(z))}{\partial y^2}+\frac{\partial^2(X(x)Y(y)Z(z))}{\partial z^2}\right)\quad(1)$$

Since $X(x)Y(y)Z(z)$ are functions of separate variables, when we take the second partial derivatives, we get:

- **For the x-direction:**

$$\frac{\partial^2}{\partial x^2}\left(X(x)Y(y)Z(z)\right) = Y(y)Z(z)\frac{d^2X(x)}{dx^2}$$

- **For the y-direction:**

$$\frac{\partial^2}{\partial y^2}\left(X(x)Y(y)Z(z)\right) = X(x)Z(z)\frac{d^2Y(y)}{dy^2}$$

- **For the z-direction:**

$$\frac{\partial^2}{\partial z^2}\left(X(x)Y(y)Z(z)\right) = X(x)Y(y)\frac{d^2Z(z)}{dz^2}$$

Next, we substitute these into Schrödinger's equation:

$$E[X(x)Y(y)Z(z)] = -\tfrac{\hbar^2}{2m}\left(Y(y)Z(z)\tfrac{d^2X(x)}{dx^2} + X(x)Z(z)\tfrac{d^2Y(y)}{dy^2} + X(x)Y(y)\tfrac{d^2Z(z)}{dz^2}\right) \tag{2}$$

Now, we divide both sides by $X(x)Y(y)Z(z)$ in order to separate the variables:

$$E = -\frac{\hbar^2}{2m}\left(\frac{1}{X(x)}\frac{d^2X(x)}{dx^2} + \frac{1}{Y(y)}\frac{d^2Y(y)}{dy^2} + \frac{1}{Z(z)}\frac{d^2Z(z)}{dz^2}\right) \tag{3}$$

Since the left-hand side involves sums of terms that depend on $x,\ y,\ z$ for the equation to hold true for all $x,\ y$ and z each term must be equal to a constant. Thus, we introduce separation constants (k_x^2, k_y^2, k_z^2) for each term:

$$\frac{1}{X(x)}\frac{d^2X(x)}{dx^2} = -k_x^2,\ \frac{1}{Y(y)}\frac{d^2Y(y)}{dy^2} = -k_y^2,\ \frac{1}{Z(z)}\frac{d^2Z(z)}{dz^2} = -k_z^2$$

Then, we get three independent ordinary differential equations:

- **For** $X(x)$:

$$\frac{d^2X(x)}{dx^2} = -k_x^2X(x)$$

- **For** $Y(y)$:

$$\frac{d^2Y(y)}{dy^2} = -k_y^2Y(y)$$

- **For** $Z(z)$:

$$\frac{d^2Z(z)}{dz^2} = -k_z^2Z(z)$$

Now, we use the fact that the total energy E is the sum of the energies corresponding to each dimension:

$$E = \frac{\hbar^2}{2m}\left(k_x^2 + k_y^2 + k_z^2\right) \tag{4}$$

IV.IV Energy Quantization Each of these terms must correspond to a set of quantum numbers n_x, n_y, n_z, giving the allowed values of k_x^2, k_y^2, k_z^2:

$$k_x = \frac{n_x \pi}{L_x}, k_y = \frac{n_y \pi}{L_y}, k_z = \frac{n_z \pi}{L_z}$$

The wave number $k = \frac{n\pi}{L}$ in the particle-in-a-box problem comes from the boundary conditions imposed on the wavefunction. This relationship is derived as follows:

1. The general solution to the Schrödinger equation for a particle in a box is of the form:
$$\psi(x) = Asin(kx) + Bcos(kx)$$
2. At $x = 0$ (one boundary of the box), $\psi(0) = 0$. This condition eliminates the cosine term, leaving:
$$\psi(x) = A\sin(kx)$$
3. At $x = L$ (the other boundary), $\psi(L) = 0$. This means:
$$A\sin(kL) = 0$$
4. For this equation to be true, $\sin(kL)$ must equal zero. This occurs when:
$$kL = n\pi$$
where n is an integer.
5. Thus, for k, we get:
$$k = \frac{n\pi}{L}$$

This relationship ensures that the wavefunction satisfies the boundary conditions of being zero at both ends of the box, which is necessary for the particle to be confined within the box. It also leads to the quantization of energy levels, as the allowed values of k directly determine the allowed energies of the particle.

Afterwards, substituting these into the energy expression (eqn. 4), we get:

$$E = \frac{\hbar^2}{2m}\left(\left(\frac{n_x \pi}{L_x}\right)^2 + \left(\frac{n_y \pi}{L_y}\right)^2 + \left(\frac{n_z \pi}{L_z}\right)^2\right)$$

This expression gives the allowed energy levels of the particle in a 3D box, where $n_x,\ n_y,\ n_z$ are positive integers ($n_x,\ n_y,\ n_z = 1, 2, 3, ...$) Thus, the energy of the particle is:

$$E_{n_x,n_y,n_z} = \frac{\hbar^2 \pi^2}{2m}\left(\frac{n_x^2}{L_x^2} + \frac{n_y^2}{L_y^2} + \frac{n_z^2}{L_z^2}\right)$$

Where L_x, L_y, L_z are the lengths of the box in the x, y and z directions, and $n_x,\ n_y,\ n_z$ are the quantum numbers corresponding to each direction.

IV.V Wavefunction Solutions The wavefunction $\psi(x, y, z)$ is the product of the individual functions $X(x)Y(y)Z(z)$, each of which is a solution to their respective 1D Schrödinger equations. For each dimension, the solutions are:

$$X(x) = \sqrt{\frac{2}{L_x}} \sin \frac{n_x \pi x}{L_x}$$

$$Y(y) = \sqrt{\frac{2}{L_y}} \sin \frac{n_y \pi y}{L_y}$$

$$Z(z) = \sqrt{\frac{2}{L_z}} \sin \frac{n_z \pi z}{L_z}$$

In conclusion, the total wavefunction is:

$$\psi(x, y, z) = \sqrt{\frac{2}{L_x}} \sin \frac{n_x \pi x}{L_x} \times \sqrt{\frac{2}{L_y}} \sin \frac{n_y \pi y}{L_y} \times \sqrt{\frac{2}{L_z}} \sin \frac{n_z \pi z}{L_z}$$

simplifying to:

$$\psi(x, y, z) = \left(\frac{2}{L}\right)^{\frac{3}{2}} \sin \frac{n_x \pi x}{L_x} \sin \frac{n_y \pi y}{L_y} \sin \frac{n_z \pi z}{L_z} \tag{5}$$

IV.VI Normalization and Probability Density The wavefunction must be normalized to ensure that the total probability of finding the particle within the box is unity. The normalization condition requires:

$$\int_0^{L_x} \int_0^{L_y} \int_0^{L_z} |\psi(x, y, z)|^2, dx, dy, dz = 1$$

The normalized wavefunction ensures that when we integrate the probability density $|\psi(x, y, z)|^2$ over the entire volume of the box, we obtain a total probability of 1.

Probability Density Visualization The probability density $|\psi(x, y, z)|^2$ provides insight into the likely locations of the particle within the box. Two particularly interesting cases are:

- **Ground State** $(n_x = 1, n_y = 1, n_z = 1)$**: This represents the lowest energy state, where the particle is most likely to be found near the center of the box, with equal probability distributions in x, y, and z directions. The ground state energy level is:** $|\psi_{1,1,1}(x, y, z)|^2 = \left(\frac{2}{L}\right)^3 \sin^2\left(\frac{\pi x}{L}\right) \sin^2\left(\frac{\pi y}{L}\right) \sin^2\left(\frac{\pi z}{L}\right)$
- **First Excited State** $(n_x = 2, n_y = 1, n_z = 1)$**: This state introduces additional nodes in the x-direction, creating regions of alternating high and low probability. The first excited state energy level is:** $|\psi_{2,1,1}(x, y, z)^2| = \left(\frac{2}{L}\right)^3 \sin^2\left(\frac{2\pi x}{L}\right) \sin^2\left(\frac{\pi y}{L}\right) \sin^2\left(\frac{\pi z}{L}\right)$

V Computational Methods

V.I Introduction The following explains how to solve the Schrödinger equation for a particle in a potential well using numerical methods. We use a finite difference approach to discretize the Schrödinger equation and solve for the eigenvalues (energy levels) and eigenvectors (wavefunctions).

V.II Setting Up The Potential Well We first define a potential well in 2D space, either in the shape of a circle, polygon, or other custom shapes such as hearts or seed patterns. The potential is represented as a matrix, where each element corresponds to a point in space.

- The potential well can be represented in several forms like polygonal, circular, bi-circular, toroidal, rose-shaped, heart-shaped, or a pattern derived from an image.
- These wells are represented as 2D matrices where each element corresponds to a point in space. The points inside the well are assigned a value **inWell**, and those outside are assigned **outWell**.

For Example:

- makeCircleWellMatrix: Creates a circular potential well by assigning inWell to points within a circle of radius (n-1)/2 and outWell to points outside.
- makeSeedWellMatrix: Generates a potential well from an image (seedmap.png). The image is read, and its binary values are used to assign inWell and outWell to the matrix elements.

V.III Hamiltonian Construction The Hamiltonian is the operator that governs the behavior of the system. For the finite potential well, the Hamiltonian is typically written as:

$$\hat{H} = -\frac{\hbar^2}{2m}\nabla^2 + V(x,y)$$

where, once again, the ∇^2 is the Laplacian operator, and $V(x,y)$ is the potential energy.

Here, the Hamiltonian matrix is constructed in a finite difference representation, where the Laplacian is approximated by the second derivative. This is done using a sparse matrix **(scipy.sparse.diags)** to represent the Hamiltonian efficiently.

V.IV Solving the Eigenvalue Problem: The Hamiltonian matrix is diagonalized using scipy.sparse.linalg.eigs, which computes the eigenvalues and eigenvectors. The eigenvalues correspond to the energy levels of the system, and the eigenvectors correspond to the wavefunctions of the system at each energy level.

```python
e_values , e_vec = eigs(Hamiltonian , k = Elevels)
```

Listing 1: Python code

This function computes the top **Elevels** eigenvalues and the corresponding eigenvectors of the Hamiltonian.

V.V Visualizing the Eigenstates: After computing the eigenvectors, the squared absolute value of the wavefunction (i.e., ψ^2) is plotted as an image to visualize the probability distribution of the particle in each state. This is done using the **imshow** function from **matplotlib**, which plots each eigenvector reshaped into a 2D grid corresponding to the spatial grid of the potential well.

```
plot = plt.imshow(pow(np.absolute(e_vec[:,i].reshape(N, N)), 2), cmap='
    nipy_spectral', interpolation='gaussian')
```

Listing 2: Python code

V.VI Saving and Displaying Results: The code saves the eigenvectors (wavefunctions) to a .npy file, which can be loaded later for analysis. It then also saves the images of the eigenfunctions as PNG files for visual inspection.

```
np.save('data_E_vectors_seed' + str(N) + 'x' + str(N) + 'e' + str(
    Elevels), e_vec)
```

Listing 3: Python code

VI Relating to Schrödinger Equation Work

VI.I Finite Potential Box The code numerically solves the Schrödinger equation for a particle in a potential well. It represents the potential well as a matrix and solves the eigenvalue problem for the Hamiltonian. The resulting energy levels and wavefunctions correspond to the quantized energy levels and wavefunctions of a particle in a potential box, similar to what was discussed earlier regarding the particle in a finite square potential well. The potential well can be shaped in various forms, including circular, polygonal, or custom patterns, providing flexibility in modeling different physical scenarios.

VI.II Energy Quantization The eigenvalues (`e_values`) obtained from solving the Hamiltonian represent the quantized energy levels, as discussed in the earlier explanation of energy quantization. These values reflect the allowed energy states of the particle inside the potential well. Just as boundary conditions were used to determine the energy levels for a finite square well, the boundary conditions in the code are set through the potential matrix, ensuring that the particle's energy is quantized.

VI.III Visualizing Wavefunctions The wavefunctions corresponding to each energy state are visualized by plotting the probability densities $|\psi(x,y)|^2$ in 2D. This is similar to the visualization of wavefunctions in simple potentials that was discussed earlier. The code produces visualizations that show the spatial distribution of the particle's probability density for different energy levels, giving insight into how the particle behaves within the potential well. These visualizations provide a clear interpretation of the quantized modes of the particle.

VI.IV Boundary Conditions The potential well matrix (referred to as `mesh` in the code) defines the boundary conditions of the system. For instance, a value of `inWell = 1` inside the well and `outWell = 0` outside the well simulates a hard wall potential. This setup mirrors the Dirichlet boundary condition $\psi(x) = 0$ at the boundaries of a finite square well, as described in the earlier discussion. The matrix representation ensures that the particle's wavefunction is confined within the potential well, adhering to the specified boundary conditions.

VI.V All the Code Listings The following are all the codes employed in my visualization.

```
    import numpy as np
import math
import matplotlib.pyplot as plt
from PIL import Image
from scipy.sparse.linalg import eigs
from scipy.sparse import diags, dia_matrix

# Number of eigenvectors (energy levels) to calculate
num_energy_levels = 50

# Image filename to define the potential
image_file = 'seedmap.png'

# Range of eigenvectors to generate images for
energy_levels_range = [0, 16]

# Size of the matrix (n x n) for the potential well
# Set n = 30 for faster computation. Larger n increases computation
    time significantly.
matrix_size = 128

# Value inside the potential well
inside_well_value = 1

# Value outside the potential well
outside_well_value = 0

# Function to create a polygonal potential well shape
def create_polygonal_shape(sides, radius=1, rotation=0, translation=
    None):
    segment_angle = math.pi * 2 / sides
    points = [
        (math.sin(segment_angle * i + rotation) * radius,
         math.cos(segment_angle * i + rotation) * radius)
        for i in range(sides)]
    if translation:
        points = [[sum(pair) for pair in zip(point, translation)]
                  for point in points]
    return points

# Function to check if a point is inside a polygon
def is_point_in_polygon(x, y, polygon_x, polygon_y):
    count = 0
    for i in range(len(polygon_x)):
        if (((polygon_y[i] <= y and y < polygon_y[i-1]) or (polygon_y[i
    -1] <= y and y < polygon_y[i])) and
            (x > (polygon_x[i-1] - polygon_x[i]) * (y - polygon_y[i]) /
     (polygon_y[i-1] - polygon_y[i]) + polygon_x[i])):
            count = 1 - count
    return count
```

```

# Function to create a polygonal potential well matrix
def generate_polygonal_well_matrix(n, inside_value, outside_value):
    well_matrix = np.empty((n, n))
    sides = 6  # Define the number of sides for the polygon
    polygon_points = create_polygonal_shape(sides, 100, 0, (100, 100))
    polygon_x = tuple([polygon_points[i][0] for i in range(0, sides)])
    polygon_y = tuple([polygon_points[i][1] for i in range(0, sides)])

    for i in range(0, n):
        for j in range(0, n):
            if is_point_in_polygon(i, j, polygon_x, polygon_y):
                well_matrix[i][j] = inside_value
            else:
                well_matrix[i][j] = outside_value
    return well_matrix

# Function to create a circular potential well matrix
def generate_circular_well_matrix(n, inside_value, outside_value):
    well_matrix = np.empty((n, n))
    for i in range(0, n):
        for j in range(0, n):
            if math.dist([i, j], [(n-1)/2, (n-1)/2]) <= (n-1)/2:
                well_matrix[i][j] = inside_value
            else:
                well_matrix[i][j] = outside_value
    return well_matrix

# Function to create a double-circle potential well matrix
def generate_bi_circular_well_matrix(n, inside_value, outside_value):
    well_matrix = np.empty((n, n))
    r = n / (2 + math.sqrt(2))
    for i in range(0, n):
        for j in range(0, n):
            if (math.dist([i, j], [r, r]) <= r) or (math.dist([i, j], [
   n - r - 3, n - r - 3]) <= r):
                well_matrix[i][j] = inside_value
            else:
                well_matrix[i][j] = outside_value
    return well_matrix

# Function to create a toroidal potential well matrix
def generate_toroidal_well_matrix(n, inside_value, outside_value):
    well_matrix = np.empty((n, n))
    r = n / (2 + math.sqrt(2))
    for i in range(0, n):
        for j in range(0, n):
            if (math.dist([i, j], [(n-1)/2, (n-1)/2]) <= (n-1)/2) and (
   math.dist([i, j], [(n-1)/2, (n-1)/2]) > (n-1)/4):
                well_matrix[i][j] = inside_value
            else:
                well_matrix[i][j] = outside_value
    return well_matrix

# Function to create a rose-shaped potential well matrix
def generate_rose_well_matrix(n, inside_value, outside_value):
    well_matrix = np.empty((n, n))
    sides = 100
    polygonA = create_polygonal_shape(sides, n/2, 0, (n/2 - 10, n/2))
    polygon_x1 = tuple([polygonA[i][0] for i in range(0, sides)])
    polygon_y1 = tuple([polygonA[i][1] for i in range(0, sides)])

    polygonB = create_polygonal_shape(sides, n/2, (2/3) * math.pi, (n
   /2- 10, n/2))
    polygon_x2 = tuple([polygonB[i][0] for i in range(0, sides)])
    polygon_y2 = tuple([polygonB[i][1] for i in range(0, sides)])

    polygonC = create_polygonal_shape(sides, n/2, -(2/3) * math.pi, (n
   /2- 10, n/2))
    polygon_x3 = tuple([polygonC[i][0] for i in range(0, sides)])
    polygon_y3 = tuple([polygonC[i][1] for i in range(0, sides)])

    for i in range(0, n):
        for j in range(0, n):
            if is_point_in_polygon(i, j, polygon_x1, polygon_y1) or
   is_point_in_polygon(i, j, polygon_x2, polygon_y2) or
   is_point_in_polygon(i, j, polygon_x3, polygon_y3):
                well_matrix[i][j] = inside_value
            else:
                well_matrix[i][j] = outside_value

    well_matrix[int(n/2 - 10)][int(n/2)] = inside_value
    well_matrix[int(n/2 - 10)][int(n/2 - 1)] = inside_value
    well_matrix[int(n/2 - 10)][int(n/2 + 1)] = inside_value

    return well_matrix

# Function to create a heart-shaped potential well matrix
def generate_heart_well_matrix(n, inside_value, outside_value):
    well_matrix = np.empty((n, n))
    for i in range(0, n):
        for j in range(0, n):
            x = (i - n/2)/110
            y = (j - n/2 + 20)/110
            if 5*((x**2 + y**2 - 1)**3) < 6*(x**2)*(y**3):
                well_matrix[i][j] = inside_value
            else:
                well_matrix[i][j] = outside_value
    return well_matrix

# Function to create a seed-shaped potential well matrix from an image
def generate_seed_well_matrix(n, inside_value, outside_value):
    well_matrix = np.empty((n, n))
    image = Image.open(image_file)
    image = np.array(image.convert('1'))
    for i in range(0, n):
        for j in range(0, n):
            if image[i][j]:
                well_matrix[i][j] = inside_value
            else:
                well_matrix[i][j] = outside_value
    return well_matrix

# General function to compute the Hamiltonian matrix for the potential
def compute_hamiltonian(matrix_2d_potential):
    flattened_potential = np.matrix.flatten(matrix_2d_potential)
```

```python
    num_points = matrix_size * matrix_size
    increment_value = -1 / (np.linspace(0, 1, matrix_size)[1] ** 2)
    zero_value = 4 / increment_value

    diagonal_matrix_N = [increment_value * flattened_potential[i] for i in range(0, num_points - matrix_size)]
    diagonal_matrix_minus_1 = [increment_value * flattened_potential[i] for i in range(0, num_points - 1)]
    diagonal_matrix_0 = [zero_value for i in range(0, num_points)]
    diagonal_matrix_plus_1 = [increment_value * flattened_potential[i] for i in range(0, num_points - 1)]
    diagonal_matrix_plus_N = [increment_value * flattened_potential[i] for i in range(0, num_points - matrix_size)]

    diagonals = [-matrix_size, -1, 0, 1, matrix_size]
    matrix_values = [diagonal_matrix_N, diagonal_matrix_minus_1, diagonal_matrix_0, diagonal_matrix_plus_1, diagonal_matrix_plus_N]

    hamiltonian_matrix = diags(matrix_values, diagonals, format='dia')
    return hamiltonian_matrix

# Function to solve for the eigenvalues and eigenvectors of the Hamiltonian matrix
def solve_hamiltonian(hamiltonian_matrix):
    eigenvalues, eigenvectors = eigs(hamiltonian_matrix, k=num_energy_levels)
    return eigenvalues, eigenvectors

# Function to display and save the images of the wavefunctions
def visualize_and_save_wavefunctions(eigenvectors):
    for i in range(int(energy_levels_range[0]), int(energy_levels_range[1])):
        fig, ax = plt.subplots(1, 1)
        plot = plt.imshow(np.abs(eigenvectors[:, i].reshape(matrix_size, matrix_size)) ** 2, cmap='nipy_spectral', interpolation='gaussian')
        plt.setp(ax, xticks=[], yticks=[])
        plt.savefig(f'{i}_wavefunction.png', bbox_inches='tight')

# Main program execution
if __name__ == '__main__':
    potential_matrix = generate_circular_well_matrix(matrix_size, inside_well_value, outside_well_value)
    plt.imshow(potential_matrix)
    plt.show()

    hamiltonian_matrix = compute_hamiltonian(potential_matrix)
    eigenvalues, eigenvectors = solve_hamiltonian(hamiltonian_matrix)

    # Sort the eigenvalues and eigenvectors
    idx = eigenvalues.argsort()[::-1]
    eigenvalues = eigenvalues[idx]
    eigenvectors = eigenvectors[:, idx]

    # Save the eigenvectors (wavefunctions)
    np.save(f'wavefunctions_{matrix_size}x{matrix_size}_e{num_energy_levels}.npy', eigenvectors)

    visualize_and_save_wavefunctions(eigenvectors)

    print('Computation and visualization complete.')
```

This wasn't the complete set of code; this is just the first part. I wrote another program that handles the following tasks:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
from matplotlib import cm
from matplotlib.colors import ListedColormap, LinearSegmentedColormap

# Load eigenvector data
eigen_vectors = np.load('data_E_vectors_sample128x128e50.npy')
grid_size = 128

print('Successfully loaded data')

# Define custom color maps
def cmap1():
    segments = 10
    total_colors = 256

    color_values = np.ones((total_colors, 4))
    color_values[:, 0] = np.array([(i % segments) / segments for i in
    range(total_colors)])
    color_values[:, 1] = np.array([((i + 5) % segments) / segments for
    i in range(total_colors)])
    color_values[:, 2] = np.array([((i + 7) % segments) / segments for
    i in range(total_colors)])
    new_color_map = ListedColormap(color_values)

    return new_color_map

def cmap2():
    color_palette = [
        (234/255, 230/255, 202/255),
        (114/255, 0, 0),
        (234/255, 230/255, 202/255),
        (114/255, 0, 0),
        (234/255, 230/255, 202/255),
        (114/255, 0, 0),
        (30/255, 23/255, 20/255),
        (234/255, 230/255, 202/255),
        (114/255, 0, 0),
        (30/255, 23/255, 20/255),
        (234/255, 230/255, 202/255),
        (30/255, 23/255, 20/255),
        (114/255, 0, 0)
    ]
    custom_cmap = LinearSegmentedColormap.from_list('color_palette',
    color_palette, N=30)
    return custom_cmap

# Generate and save images for eigenvectors 1 through 49
for index in range(1, 50):
    figure(num=None, figsize=(6, 6), dpi=300)
    plt.axis('off')  # Hide axis

    # Reshape the eigenvector and compute the probability density
    probability_density = np.abs(eigen_vectors[:, index].reshape(
   grid_size, grid_size)) ** 2

    # Select color map
    selected_cmap = 'nipy_spectral'

    # Plot the data
    plot_image = plt.imshow(probability_density, cmap=selected_cmap,
   interpolation='lanczos')

    # Save the image
    plt.savefig(f'Image_{index}_result.png', bbox_inches='tight')
    print(f'Image {index} saved successfully')

    # Close the plot to release memory
    plt.close()

print('All images have been saved successfully')
```

VI.VI What the code does: When I developed this script, my primary goal was to transform complex quantum mechanical eigenvector data into visually meaningful representations. The challenge was to take abstract numerical data and create intuitive images that could reveal the underlying structural patterns of quantum states.

I first started by loading a pre-computed eigenvector dataset from a NumPy file

```
    (data_E_vectors_seed128x128e50.npy)
```

This file contained 50 eigenvectors, each representing a unique quantum state configuration for a 128x128 grid.

One of my key design choices was implementing custom color mapping functions. I created two unique color mapping strategies:

- *cMap1():* This function creates a cyclical color modulation technique. By carefully manipulating red, green, and blue components through periodic cycles, I could generate a dynamic color palette that emphasizes different characteristics of the eigenvectors.
- *cMap2():* Here, I crafted a more deliberate color palette using custom RGB values. I intentionally mixed neutral tones (like sandy beiges) with deep reds and dark browns to create a nuanced color scheme that could highlight subtle variations in the probability densities.

The core visualization process involves several critical steps. For each eigenvector, I:

- Reshaped the 1D vector into a 2D matrix matching the original grid
- Computed the squared absolute value to represent probability density
- Visualized the data using high-resolution image generation

I chose to use the

```
    'nipy_spectral'
```

colormap for most visualizations, though I kept cMap2() as an alternative. The spectral colormap provides excellent contrast and is particularly effective for scientific visualizations. Performance and memory management were also key considerations. By using plt.close() after each image generation and configuring tight bounding boxes, I ensured the script could efficiently process multiple eigenvectors without consuming excessive computational resources.

The output is a series of PNG images, each representing a different quantum state's probability density. These images are not just visually interesting—they provide a powerful way to understand the spatial distribution of quantum states across different energy configurations. Ultimately, this script bridges the gap between complex mathematical data and visual intuition.

VII Results

Visualization of Quantum Mechanical Eigenvectors The following 4 images visualizes the probability densities of quantum mechanical eigenvectors. These images represent the spatial distribution of the quantum states and offer insight into the wave function's behavior across different energy levels.

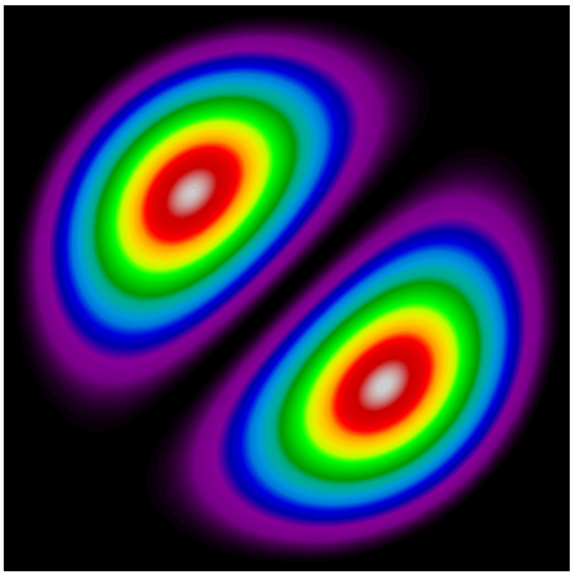
Figure 1: First Excited State (2, 1, 0)

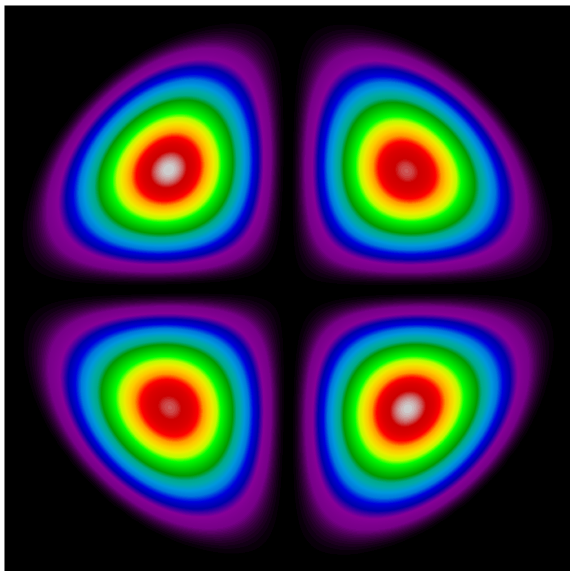
Figure 2: Third Excited State (3, 2, 1)

Figure 1 represents the first excited state of the quantum system. It shows a more complex spatial distribution of the wave function compared to the fundamental state, indicating the system's first level of energy excitation. On the other hand, Figure 2 depicts the third excited state of the quantum system, showcasing an even more intricate distribution of wave function probability, with additional nodes and regions of higher probability density as the energy level increases.

Here's two more intricate patterns that the hydrogen atom starts to exhibit.

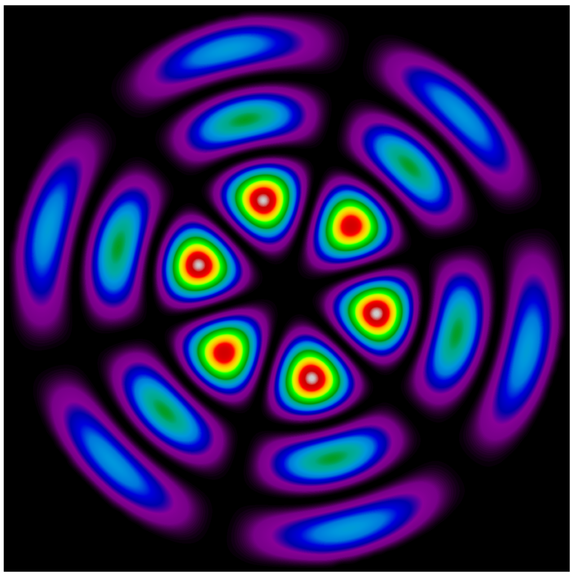
Figure 3: n = 35

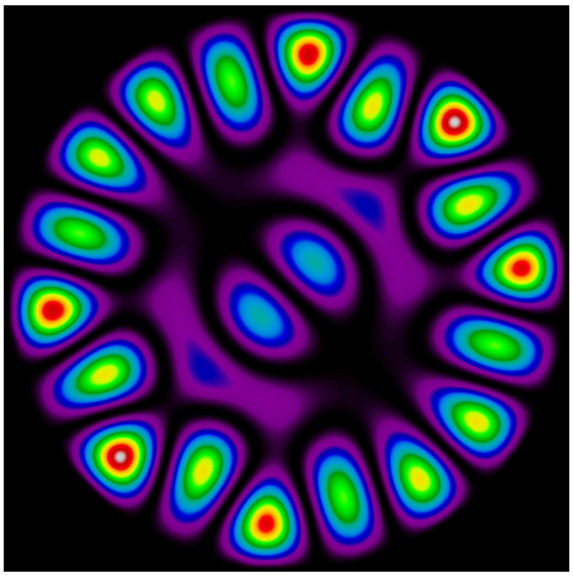
Figure 4: n = 39

Additional visualizations and images related to the quantum mechanical eigenvectors can be found in the supplementary materials on my GitHub repository. The program used to generate these images is also available, allowing readers to replicate the results and run the simulations independently.

VIII Conclusion

This project, undertaken as part of my honors course in differential equations, allowed me to gain a deeper understanding of quantum mechanics and computational modeling. By developing heatmaps and a 3D model to analyze Schrödinger's wave equation using hydrogen as the base particle, I significantly improved my skills in solving differential equations, Python programming, and data visualization. The process of solving the time-independent Schrödinger equation for a particle confined in a finite potential well not only deepened my theoretical knowledge but also provided valuable hands-on experience with computational techniques. The challenge of analyzing the quantized energy levels and visualizing the wave function's probability densities brought new insights into how quantum systems behave under realistic conditions, as opposed to idealized models like the infinite potential box.

This project was particularly valuable because it allowed me to apply the concepts learned in my differential equations course in a practical, real-world context. The experience of solving the Schrödinger equation and interpreting the results through visualizations was both intellectually rewarding and technically enriching. It enhanced my appreciation for the power of computational modeling in understanding complex physical systems.

I would like to express my sincere gratitude to my mentor, Kenyatta Weathersby, for allowing me and giving me the opportunity to do this project.

Credits

- Greg Egan - Symmetric Waves
- Ohio State University - Phase and Orbitals
- GitHub - Hydrogen Wavefunctions by ssebastianmag
- University of Illinois - Electromagnetic Theory (PHYS480)
- Wikipedia - Laplace Operator
- LibreTexts - The Schrödinger Equation and a Particle in a Box
- Wikipedia - Schrödinger Equation
- University of Washington - Physics 227 Reading on Quantum Mechanics
- LibreTexts - Particle in a 1-Dimensional Box
- LibreTexts - Energy Quantization in a Particle in a Box
- OpenStax - University Physics Volume 3: The Quantum Particle in a Box
- EITCA - Particle in a Box Boundary Conditions and Quantization

Finally, I would like to acknowledge Skyline College for providing the resources and facilities necessary for this project.