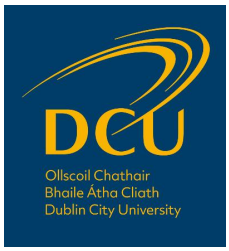


Final Year Project Report

Quantum 2-electron wavefunctions: Calculation and Visualization

Name: Adam Morrissey

Student No.: 19359553

Class: AP4

Date: 11th April 2023

Supervisor: Dr. Lampros Nikolopoulos

Declaration

Name: Adam Morrissey
Student ID Number: 19359553
Programme: AP4
Module Code: PS451
Assignment Title: Final year project
Submission Date: 11th April 2023

I understand that the University regards breaches of academic integrity and plagiarism as grave and serious.

I have read and understood the DCU Academic Integrity and Plagiarism Policy. I accept the penalties that may be imposed should I engage in practice or practices that breach this policy.

I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations, paraphrasing, discussion of ideas from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the sources cited are identified in the assignment references.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

I have used the DCU library referencing guidelines (available at: http://www.library.dcu.ie/LibraryGuides/Citing&ReferencingGuide/player.html) and/or the appropriate referencing system recommended in the assignment guidelines and/or programme documentation.

By signing this form or by submitting this material online I confirm that this assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

By signing this form or by submitting material for assessment online I confirm that I have read and understood DCU Academic Integrity and Plagiarism Policy (available at: http://www.dcu.ie/registry/examinations/index.shtml).

Name: Adam Morrissey

Date: 10th April 2023

Abstract

This report explores the numerical solutions of the Schrödinger equation for two electrons using finite difference methods. Specifically, the time-dependent Schrödinger equation is solved using various finite difference schemes to study the scattering of interacting Gaussian wavepackets in one and two dimensions. The results of the simulations are visually demonstrated through plots and animations, providing a clearer understanding of the behavior of two electrons and their interactions in quantum mechanics.

Acknowledgements

Above all, I would like to express my gratitude to Dr Lampros Nikolopoulos, my supervisor, for entrusting me with the responsibility to work on this project and for broadening my horizons beyond the technical aspects of the task. Additionally, I am deeply indebted to my family and friends for their unwavering support and motivation that have been the driving force behind my progress throughout the year

Contents

List of Figures

List of Tables

Introduction

The Schrödinger equation, is a linear partial differential equation, which dictates the evolution of quantum states in a given system. It is used as a mathematical framework for understanding the wave-like behavior of particles at microscopic scales. We can draw a parallel to Newton's second law, which predicts the future coordinates of a body based on its initial coordinates and the forces that act on it, the Schrödinger equation does something similar but instead it describes the evolution of probability waves that determine the motion of particles within a specific field. However, instead of providing an exact position, the Schrödinger equation predicts the probability distribution of a particle in space, shedding light on the inherently probabilistic nature of quantum systems.

While analytical solutions to the Schrodinger equation exist for simple quantum mechanical systems, such as the hydrogen atom where a single electron and a nucleus are present, the Time-Dependent Schrödinger Equation (TDSE) cannot be solved analytically for more complex multi-electron systems. For these cases, researchers need to resort to numerical methods and computational techniques to approximate solutions to give insight into the behavior of these systems. The main reason the two electron Schrodinger exemplifies this problem, is due to the presence of electron-electron interactions (repulsive Coulomb potential) which complicate the mathematical description of the system. The mutual electrostatic repulsion between the negatively charged electrons bring another layer of issues to the equation, requiring very advanced techniques and approximations beyond the scope of this report for obtaining meaningful results. These approaches, such as the Hartree-Fock method, density function theory and configuration interaction, have been developed to tackle the challenges posed by the two electron Schrödinger equation and other multi-electron systems.

The two electron form of the Schrödinger equation is:

$$i\hbar\frac{\partial}{\partial t}\Psi(r_1, r_2, t) = [-\frac{\hbar^2}{2m}\frac{d^2}{dr_1^2} + \frac{\hbar^2}{2m}\frac{d^2}{dr_2^2} + \frac{e^2}{|r_1 - r_2|}]\Psi(r_1, r_2, t) \tag{1.1}$$

We can express Eqn 1.1 in a more simplified manner, by representing them

as two time dependent particles with some potential:

$$\begin{aligned} i\frac{\partial}{\partial t}\psi(r_1, r_2, t) &= H\psi(r_1, r_2, t) \\ H &= -\frac{1}{2m_1}\frac{\partial^2}{\partial r_1^2} - \frac{1}{2m_2}\frac{\partial^2}{\partial r_1^2} + V(r_1, r_2) \end{aligned} \tag{1.2}$$

H represents the Hamiltonian operator, while m_i and x_i represent the mass and position of particle $i = 1, 2$. To make things easier, we've set $\hbar = 1$. We can learn a lot about the quantum system by calculating the two-particle wave function, $\psi(r_1, r_2, t)$.

The two-particle wave function, in particular, allows us to compute the probability density of particle 1 being at r_1 and particle 2 being at r_2 at a given time t. To calculate this probability density, we can use the following definition:

$$\rho(r_1, r_2, t) = |\psi(r_1, r_2, t)|^2 \tag{1.3}$$

By examining the probability density, we can acquire a deeper understanding of the spatial distribution and dynamic behavior of the particles within the quantum system. However, before proceeding with the calculation of the probability density, it is crucial to acknowledge a key constraint: both particle 1 and particle 2 must be located somewhere in space at any given moment. This fact necessitates a normalization constraint on the wave function, ensuring that the total probability of finding the particles within the defined spatial domain equals one at every instant in time.

The normalization constraint can be mathematically expressed as:

$$P(t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} dr_1 dr_2 \, |\psi(r_1, r_2, t)|^2 = 1 \tag{1.4}$$

The reduction of the two-body wave function into a single-particle wave function, $\psi(r)$, is generally an approximation, unless the system is uncorrelated. In an uncorrelated system, the total wave function can be expressed in a product form, signifying that the particles do not interact with each other. Nevertheless, even in correlated systems, it is possible to extract meaningful one-particle densities from the two-particle density by integrating over the other particle's coordinates.

To obtain the one-particle density for particle 1, we can integrate the two-particle probability density over the position of particle 2:

$$\rho(r_1, t) = \int_{-\infty}^{\infty} |\psi(r_1, r_2, t)|^2 \, dr_2 \tag{1.5}$$

Similarly, for particle 2, the one-particle density can be computed by integrating over the position of particle 1:

$$\rho(r_2, t) = \int_{-\infty}^{\infty} |\psi(r_1, r_2, t)|^2 \, dr_1 \tag{1.6}$$

These one-particle densities provide insights into the individual behavior of each particle, even in the presence of correlations. Naturally, the complete solution is given by the two-particle probability density $\rho(r_1, r_2, t)$. However, extracting physical insights from a function that depends on three variables can be a daunting task. As a result, it is often convenient to consider the one-particle densities $\rho_1(r_1, t)$ and $\rho_2(r_2, t)$ separately, which can be interpreted as two distinct wave packets colliding.

For identical particles, such as electrons or identical atoms, the total wave function must be either symmetric or antisymmetric under the interchange of the particles. In other words, the wave function should remain unchanged or change its sign when the particles' positions are swapped. To satisfy this requirement, we can impose the appropriate symmetry condition on the numerically-determined spatial wave function $\psi(r_1, r_2)$ by constructing the following combinations:

- Symmetric Wave function:

$$\psi_S(r_1, r_2) = \frac{1}{\sqrt{2}} \left[\psi(r_1, r_2) + \psi(r_2, r_1)\right] \tag{1.7}$$

- Antisymmetric Wave function:

$$\psi_A(r_1, r_2) = \frac{1}{\sqrt{2}} \left[\psi(r_1, r_2) - \psi(r_2, r_1)\right] \tag{1.8}$$

These combinations ensure that the wave function satisfies the symmetry or antisymmetry condition, providing a more accurate representation of the physical behavior of identical particles in a quantum system.

1.1 Aims and Objectives

The primary aim of this project is to investigate the scattering phenomena of interacting particles by numerically solving the time-dependent Schrödinger equation (TDSE) for one and two electrons. Various finite difference methods are employed to achieve this objective, while visualizations are used to enhance the understanding of quantum interactions. The specific objectives are as follows:

- Study one-dimensional wavepacket-wavepacket scattering events by implementing the Visscherr method
- Employ an Alternating Direction Implicit (ADI) technique adapted from student report and extend for the two electron Schrodinger equation
- Calculate and plot for the radial probability distribution using variational methods and trial wave functions

Theory

2.1 Finite Differences

2.1.1 Approximation Derivatives

In situations other than the most elementary ones, analytical methods often struggle to provide solutions for intricate differential equations. As a result, numerical methods are employed to derive approximate solutions. Among these methods, the finite difference method (FDM) is the most basic and frequently the most intuitive. The FDM operates by splitting the domain, where the solution can exist, into a finite number of distinct points and replacing continuous derivatives with algebraic approximations.

The error that arises when transitioning from a continuous to a finite domain is a result of using algebraic approximations instead of continuous derivatives. By employing the Taylor series, these approximations and the errors associated with them can be calculated by expanding a function with a finite difference.

Assume a function $f(x)$ has an unknown value. To estimate its derivative, we expand the Taylor series about x using a finite distance Δx:

$$f(x + \Delta x) \approx f(x) + \Delta x \frac{df(x)}{dx} + \frac{\Delta x^2}{2} \frac{d^2 f(x)}{dx^2} + \frac{\Delta x^3}{6} \frac{df^3(x)}{dx^3} ... \tag{2.1}$$

Rearranging the terms and disregarding the smallest one, we obtain:

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x} - \frac{\Delta x}{2} f''(x) \tag{2.2}$$

This expression denotes the forward difference approximation of the first derivative. It is considered first-order accurate since the primary error remaining after truncating the Taylor series is proportional to Δx. The backward difference approximation can be derived by similarly expanding in the opposite direction:

$$f'(x) = \frac{f(x - \Delta x) - f(x)}{\Delta x} - \frac{\Delta x}{2} f''(x) \tag{2.3}$$

Higher-order derivatives can be determined by expanding the Taylor series to different degrees and directions, and then combining and eliminating errors. For now, we will only present the following approximations, which will be utilized in this report:

$$f''(x) = \frac{f(x + \Delta x) - 2f(x) + f(x - \Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x^2) \tag{2.4}$$

$$f''(x) = \frac{16f(x + \Delta x) - f(x + 2\Delta x) - 30f(x) - f(x - 2\Delta x) + 16f(x - \Delta x)}{12\Delta x^2} \tag{2.5}$$

Here, $\mathcal{O}$ represents the leading error term in the Taylor series expansion.

2.1.2 Discretization of space and time

To solve the two-particle Schrödinger equation (Eq. 1.1) using the finite difference method, we must first compute the dependent variable ψ on a grid of discrete values for the independent variables, as suggested by Goldberg et al. (1967) [4]. The discretization of our domain encompasses both spatial and temporal dimensions. This section aims to demonstrate the construction of this domain and the notation employed throughout this report.

Consider the standard time dependent Schrödinger equation and let x_0 and x_{N_x} be the boundary points of our domain along the x-axis. We divide the interval $[x_0,\ x_{N_x}]$ into a uniform grid of N_x points with equal spacing Δx between them. Mathematically, we can represent the step size Δx as:

$$\Delta x = x_{i+1}, \text{where i} \in (1, 2, 3, ...,\text{N}_x\text{-1}) \tag{2.6}$$

Similarly, for the y-axis, we establish a uniform grid with boundary points y_0 and y_{N_y} The interval $[y_0,\ y_{N_y}]$ is divided into N_x points separated by step sizes Δy. The spacing Δy can be defined as:

$$\Delta y = y_{i+1}, \text{where i} \in (1, 2, 3, ...,\text{N}_y\text{-1}) \tag{2.7}$$

Additionally, for the two-particle Schrödinger equation, let Δt be the time step and $t_n = n\Delta t$, where $n \in (0, N_t)$. This allows us to express the wave function at any given point as $\psi(x_i, y_j, t_n)$. To simplify the notation, we

denote $\psi_{i,j}^n = \psi(x_i, y_j, t_n)$, which leads to the following finite difference approximations for the partial derivatives:

$$\begin{aligned}
\frac{\partial \psi}{\partial t} &\approx \frac{\psi_{i,j}^{n+1} - \psi_{i,j}^n}{\Delta t} + \mathcal{O}(\Delta t) \\
\frac{\partial^2 \psi}{\partial x^2} &\approx (\Delta x^2)^2 \equiv \frac{\psi_{i+1,j}^n - 2\psi_{i,j}^n + \psi_{i-1,j}^n}{\Delta x^2} + \mathcal{O}(\Delta x^2) \\
\frac{\partial^2 \psi}{\partial y^2} &\approx (\Delta y^2)^2 \equiv \frac{\psi_{i,j+1}^n - 2\psi_{i,j}^n + \psi_{i,j-1}^n}{\Delta y^2} + \mathcal{O}(\Delta y^2)
\end{aligned} \tag{2.8}$$

Here, $\mathcal{O}(\Delta t), \mathcal{O}(\Delta x^2)$ and $\mathcal{O}(\Delta y^2)$ represent the error terms corresponding to the time and spatial discretization, respectively.

Using these finite difference approximations, we can now reformulate the two-particle Schrödinger equation as a set of discrete equations involving the wave function $\psi_{i,j}^n$ at each point in the grid. This discretized representation enables us to solve the equation numerically, providing approximate solutions for the wave function and corresponding energy eigenvalues of the two-particle system.

Applying this theory to the two particle Schrodinger equation leads to the following grid of discrete values:

$$\psi(r_1, r_2, t) = \psi(r_1 = l\Delta r_1, r_2 = m\Delta r_2, t = n\Delta t) \equiv \psi_{l,m}^n \tag{2.9}$$

Here, l, m, and n are integers representing the indices for the discretized spatial and temporal variables. This representation allows us to approximate the continuous wave function, ψ, using a finite set of discrete values.

For each time step n, we compute the spatial part of the wave function across the entire two-dimensional grid formed by r_1 and r_2 values. Afterward, we advance the time by one step (Δt), and repeat the process to obtain the wave function over the spatial grid for the new time step. By iteratively solving for the spatial part of the wave function at each time step, we can construct an approximate solution for the two-particle Schrödinger equation throughout the domain. This numerical approach facilitates the analysis of the system's behavior and the extraction of valuable information, such as energy eigenvalues and eigenstates.

Building upon the finite difference method, we can further analyze the spatial part of the two-particle Schrödinger equation by employing Taylor expansions for the wave function, $\psi(r_1, r_2, t)$, with respect to the positions of both particles up to an accuracy of $\mathcal{O}(\Delta r^4)$. The Taylor series approximates the wave function's spatial derivatives in terms of discrete grid points, which can be used to solve the equation numerically. For example:

$$\frac{\partial^2 \psi}{\partial r_1^2} \simeq \frac{\psi(r_1 + \Delta r_1, r_2) - 2\psi(r_1, r_2) + \psi(r_1 - \Delta r_1, r_2)}{\Delta r_1^2} + \mathcal{O}(\Delta r_1^2) \quad (2.10)$$

In this context, the fourth-order accuracy in the spatial derivatives ensures that the resulting numerical solution is reasonably accurate while maintaining computational efficiency. By incorporating the Taylor expansion, we can derive finite difference approximations for the second derivatives of the wave function with respect to the positions of both particles. These approximations can then be substituted into the two-particle Schrödinger equation, replacing the continuous spatial derivatives with their corresponding discrete counterparts.

Consequently, this enables us to transform the continuous partial differential equation into a set of algebraic equations that can be solved iteratively over the discretized spatial and temporal grid. This approach, which leverages the Taylor expansion up to $\mathcal{O}(\Delta r^4)$, provides an effective means for solving the two-particle Schrödinger equation numerically, enabling the investigation of complex quantum systems that would be otherwise intractable using analytical methods.
Therefore, we can now represent the right hand side of simplified Schrödinger (Eqn 1.2) in discrete notation, it is now:

$$H\psi = -\frac{\psi_{l+1,m} - 2\psi_{l,m} + \psi_{l-1,m}}{2m_1 \Delta r_1^2} - \frac{\psi_{l,m+1} - 2\psi_{l,m} + \psi_{l,m-1}}{2m_2 \Delta r_2^2} + V_{l,m}\psi_{l,m} \quad (2.11)$$

2.2 Time Propagation Methods

As stated in the previous section, we can solve for the two particle Schrodinger equation via the finite difference method, this technique turns the partial differential equation into a set of simultaneous algebraic equations. We can now use a few different methods to evolve the TDSE in time and find future solutions from an initial wave function $\psi(r_1, r_2, t = 0)$

2.2.1 Explicit Methods

One approach to solving the problem is to use an explicit method that directly computes the wave function ψ_l, m^{n+1}. l,m is based on value at the current time step ψ_i^n. We converted the R.H.S of Eq 1.1 into a finite difference equation which involved changing the derivative in the Hamiltonian with appropriate difference equations. In the case of one-dimensional systems, the difference equations can be used to approximate the derivative of the wave function with respect to position. This enables us to compute the wave function at discrete points in space and time, making it possible to simulate the behavior of quantum systems numerically. By using an explicit method, we can propagate the solution in time using only the current value of the wave function, without the need to solve additional equations or perform matrix inversions.

The steps to numerically propagate for future time steps ψ^{n+1} involves replacing the LHS of Eq 1.1 with a first order accurate difference equation, where:

$$i\frac{\partial\psi}{\partial t} = i\frac{\psi_{l,m}^{n+1} - \psi_{l,m}^{n}}{\Delta t} \tag{2.12}$$

Now, if we combine the equations 2.12 and 2.11, we get the following:

$$i\frac{\psi_{l,m}^{n+1} - \psi_{l,m}^{n}}{\Delta t} = -\frac{\psi_{l+1,m} - 2\psi_{l,m} + \psi_{l-1,m}}{2m_1\Delta r_1^2} - \frac{\psi_{l,m+1} - 2\psi_{l,m} + \psi_{l,m-1}}{2m_2\Delta r_2^2} + V_{l,m}\psi_{l,m} \tag{2.13}$$

As previously mentioned, the index $l \in [1, N_{r_1}]$ and $m \in [1, N_{r_2}]$ represents the spatial grid points and n represents the current time step. In this context, we assume that the value of $\psi_{l,m}^n$ at the current time step is known either from a previous time step or from the initial conditions. With this assumption, the only unknown variable in the equation is the wave function at the next time step, $\psi_{l,m}^{n+1}$. By performing some algebraic manipulation, we can rearrange Eq. 2.13 and solve for $\psi_{l,m}^{n+1}$ each time step:

$$\psi_{l,m}^{n+1} = \psi_{l,m}^{n} - i\Delta t\left(-\frac{\psi_{l+1,m} - 2\psi_{l,m} + \psi_{l-1,m}}{2m_1\Delta r_1^2} - \frac{\psi_{l,m+1} - 2\psi_{l,m} + \psi_{l,m-1}}{2m_2\Delta r_2^2} + V_{l,m}\psi_{l,m}\right) \tag{2.14}$$

We can reduce this equation into a more compact form, which resembles a forward difference approximation for the time evolution operator:

$$\psi_{l,m}^{n+1} = e^{-i\Delta t H}\psi_{l,m}^{n} \simeq (1 - i\Delta t H)\psi_{l,m}^{n} \tag{2.15}$$

While this equation is straightforward and provides a simple means of computing solutions at future time steps, it has some limitations. Specifically, this explicit method is neither stable nor unitary, since due to the growing modulus of the term multiplying ψ, which increases with each time step. Specifically, the issue arises from the eigenvalue $(1 - iE\Delta t)$ which leads to the modulus $\sqrt{1 + E^2\Delta t^2}$ [1][11].

This means that errors can accumulate over time, leading to inaccurate or potentially incorrect, unstable results. To address this issue, various alternative numerical methods have been developed that aim to improve the stability and accuracy of the solution. For example, implicit methods involve solving an equation that includes the wave function at the next time step, rather than using the wave function at the current time step to directly compute the solution. This can lead to more stable solutions, but at the cost of increased computational complexity.

Semi-implicit methods, on the other hand, aim to strike a balance between stability and computational efficiency. These methods use a combination of explicit and implicit approaches, where certain parts of the equation are treated explicitly, while others are treated implicitly. This allows for more stable solutions while still maintaining reasonable computational efficiency. A number of different numerical methods have been developed to address the stability issues associated with the explicit method. For example, an improvement by Askar and Cakmak [1], which is a central difference algorithm based on the Eq 2.15:

$$\psi_{l,m}^{n+1} - \psi_{l,m}^{n-1} = (e^{-}i\Delta tH - e^{i\Delta tH})\psi_{l,m}^{n} \simeq -2i\Delta tH\psi_{l,m}^{n} \tag{2.16}$$

$$\begin{aligned} \Rightarrow \psi_{l,m}^{n+1} \simeq \psi_{l,m}^{n-1} - 2i[&\left\{(\frac{1}{m_1} + \frac{1}{m_2})4\lambda + \Delta r V_{l,m}\right\}\psi_{l,m}^{n} \\ &-\lambda\left\{\frac{1}{m_1}(\psi_{l+1,m}^{n} + \psi_{l-1,m}^{n}) + \frac{1}{m_2}(\psi_{l,m+1}^{n} + \psi_{l,m-1}^{n})\right\}] \end{aligned} \tag{2.17}$$

In this solution, we make the assumption that $\Delta r_1 = \Delta r_2$, this leads to the relationship where:

$$\lambda = \frac{\Delta t^2}{\Delta r^2} \tag{2.18}$$

This ratio is often referred to as the Courant–Friedrichs–Lewy (CFL) condition, and it plays a crucial role in determining the stability of numerical methods for solving partial differential equations, such as the Schrödinger equation. The CFL condition specifies that the time step size must be chosen such that information cannot propagate faster than a single spatial step over the course of one time step. If the time step size is too large, information can propagate faster than this limit, leading to instability and potentially inaccurate results. Therefore, in practice, the time step size is often chosen to satisfy the CFL condition to ensure stability and accuracy.

Now that we have an explicit solution Eq 2.17 where the wave function is solved for one future time based on the wavefunction of two past times. This method is computationally efficient, as it only requires storing two sets of wave function values in memory and can be iteratively applied to calculate the wave function at all future times. However, this method can suffer from stability issues and may produce inaccurate results if the time step size is not carefully chosen.

2.2.2 Implicit Methods

The implicit method is an alternative approach to solving the Schrödinger equation that eliminates the stability issues associated with explicit methods. In contrast to explicit methods, which use the current or past nodal points to find solutions for future time steps, implicit methods solve a coupled set of linear equations with multiple unknowns to find solutions to future time steps. While this approach is more computationally demanding, as it involves solving a large system of simultaneous equations, it is often unconditionally stable or has weak stability requirements to ensure the solution remains numerically bounded. This allows for larger step sizes to be used to propagate the solution, although step sizes must still be balanced in accordance with the desired accuracy.

To implement the implicit method, the spatial derivatives of the right-hand side of the Schrödinger equation are evaluated at future time steps, rather than the present ones as in the explicit method. The resulting equation is a set of simultaneous algebraic equations that can be represented in matrix form as a tridiagonal matrix. This matrix can be efficiently solved using specialized tridiagonal matrix solvers. The implicit solution for our Schrodinger

equation takes the form:

$$i\frac{\psi_{l,m}^{n+1}-\psi_{l,m}^{n}}{\Delta t}=-\frac{\psi_{l+1,m}^{n+1}-2\psi_{l,m}^{n+1}+\psi_{l-1,m}^{n+1}}{2m_1\Delta r_1^2}-\frac{\psi_{l,m+1}^{n+1}-2\psi_{l,m}^{n+1}+\psi_{l,m-1}^{n+1}}{2m_2\Delta r_2^2}+V_{l,m}^{n+1}\psi_{l,m} \tag{2.19}$$

The implicit method is often referred to as the Crank-Nicolson scheme and is represented by a tridiagonal set of linear equations. The method is unconditionally stable and accurate to the second order in time step, making it a powerful tool for solving complex systems in quantum mechanics. However, the computational cost of the implicit method is still large, as the set of linear equations must be solved at each time step.

To update wave functions at the boundaries at each moment, the Schrödinger equation with the difference approximations is solved shown in Eq 2.19, similar to what is done in the explicit method. Overall, the choice of numerical method depends on the specific problem being solved and the desired level of accuracy and computational efficiency.

We can get a more compact version of Eq 2.19, by rewriting in this format:

$$\psi_{l,m}^{n}=(1+i\Delta tH)\psi_{l,m}^{n+1} \tag{2.20}$$

This implicit solution may be more direct to compute than the explicit solution, but it requires a large amount of matrices that must be solved at each time step.

2.3 Atomic Potentials

In quantum mechanics, the atomic potential describes the interaction between particles in a system, and is an essential component of the Hamiltonian, which takes the general form of the sum of the kinetic energy operator and the potential energy operator.

$$\hat{H}=\hat{T}+\hat{V} \tag{2.21}$$

For a two-electron atomic system, the Hamiltonian is typically modeled as a sum of kinetic energy and potential energy terms, given by:

$$\hat{H}=-\frac{\hbar}{2m}(\nabla_1^2+\nabla_2^2)-\frac{e^2}{4\pi\epsilon_0}(\frac{Z}{r_1}+\frac{Z}{r_2}-\frac{1}{|r_{1,2}|}), \tag{2.22}$$

where $\hat{T}$ represents the kinetic energy operator, and $\hat{V}$ represents the potential energy operator. The first term in the Hamiltonian represents the kinetic energy of the electrons, while the second term represents the potential energy due to the Coulomb interaction between the electrons and the atomic nucleus.

In the case of a hydrogenic atom, the central potential $U_l(r)$ that governs the radial motion of an electron can be modeled fairly well by the expression:

$$U_l(r) = -\frac{Ze}{r} + \frac{l(l+1)}{2r^2}, \quad 0 < r < \infty, \tag{2.23}$$

where Ze is the atomic number and l is the quantum number characterizing the eigenvalues of the squared angular momentum operator $\hat{l}^2$ (in units of $\hbar^2$). The behavior of the radial solutions for this general atomic potential can be solved in the next section.

The nature of the electron's motion, whether it is bound or unbound, depends critically on the sign of its energy (see 2.1). For unbound states, the classically allowed region is shown, which describes the region where the electron can move freely without being bound to the atomic nucleus. However, for multi-electron atomic systems, this model is insufficient and the calculation of the corresponding effective potential $U_l(r)$ is more involved, as the electrons interact with each other and not just with the atomic nucleus.

To study the behavior of such systems, numerical methods such as the Hartree-Fock method and density functional theory are often used. These methods involve solving the Schrödinger equation for the electrons in the presence of an effective potential that takes into account the electron-electron interactions. The resulting wave functions provide information about the probability distributions of the electrons in the system, and can be used to calculate various physical properties such as energy levels, bond lengths, and reaction rates.

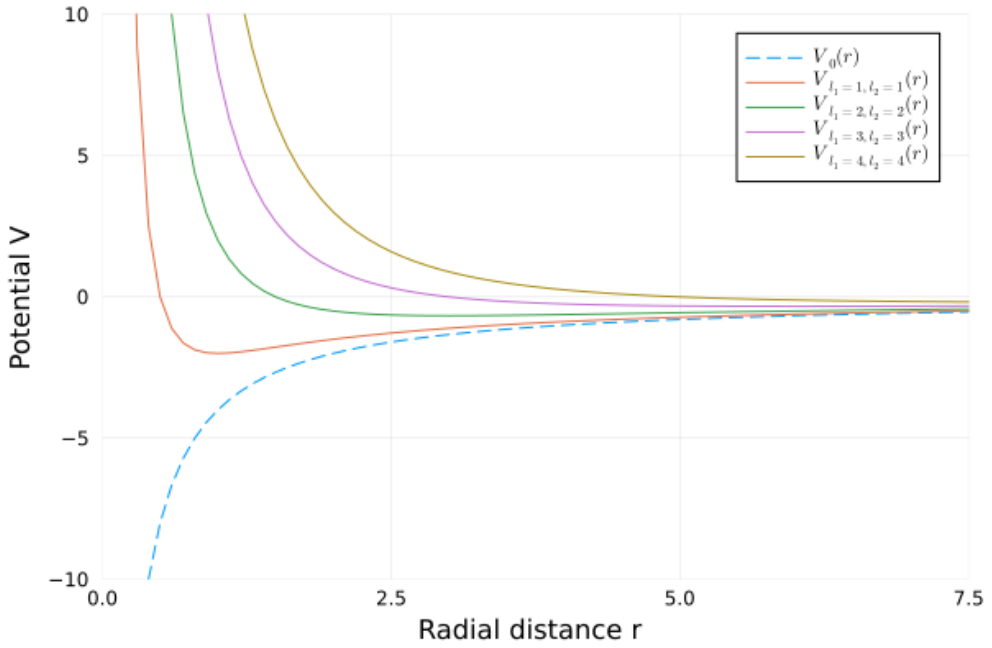


Figure 2.1: Typical atomic (radial) potential for two electron in a state of well-defined angular momentum. Here Z = 2

2.4 Radial Probability Distribution and Ground State calculation

The radial probability distribution function represents an essential concept in quantum mechanics for understanding the spatial distribution of electrons in atoms. For the helium atom, it provides valuable insights into the behavior and distribution of electrons around the nucleus. In this discussion, I'll provide an overview of the radial probability distribution for helium, and other two electron atoms.

The helium atom consists of two electrons surrounding a nucleus with two protons and two neutrons, as represented in 2.2. To describe the spatial distribution of the electrons, we utilize the concept of the radial probability distribution function, which represents the probability of finding an electron within a specific radial distance from the nucleus. The radial probability distribution function is derived from the square of the radial wave function,

which in turn is obtained from the Schrödinger equation.

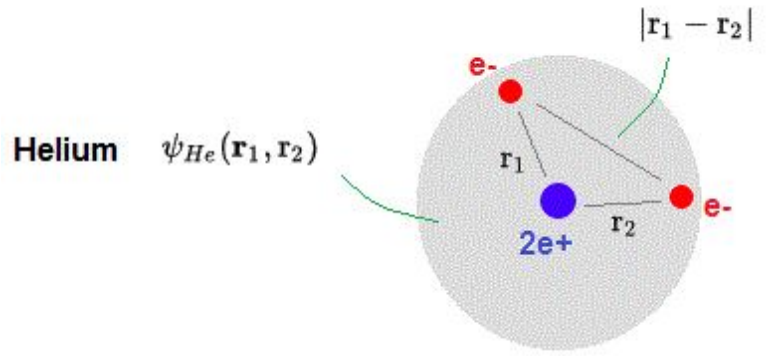


Figure 2.2: Model of Helium atom with radial distances

One of the earliest successful approximations for the helium wave function is the Hartree-Fock method, proposed by Douglas Hartree and Vladimir Fock in the 1930s [5][3]. The Hartree-Fock method is an iterative approach that accounts for electron-electron interactions and provides a more accurate description of the helium atom compared to simpler hydrogen-like approximations.

Another important contribution to understanding the helium atom is the work of Hylleraas (1929)[7], who introduced the use of wave functions containing explicit electron-electron distances to capture the correlation effects between the electrons. The Hylleraas wave function paved the way for modern correlated wavefunction methods, such as the configuration interaction (CI) and coupled-cluster (CC) approaches, which provide highly accurate descriptions of the helium atom's electronic structure.

The radial probability distribution for the helium atom can be calculated using these advanced wave function methods. These distributions provide insights into the electron density around the nucleus and the average electron-electron distances in the atom. In particular, the radial probability distribution function reveals the presence of two distinct electron shells in the helium atom. The inner shell, closer to the nucleus, corresponds to the 1s orbital,

while the outer shell represents the higher-energy 2s and 2p orbitals.

Early methods, such as the Hartree-Fock approach and the Hylleraas wave function, have provided valuable insights into the helium atom's electronic structure, while modern correlated wave function methods, like CI and CC, offer highly accurate descriptions of the radial probability distributions. These distributions offer critical information about electron density, electron-electron distances, and the spatial arrangement of electron orbitals in the helium atom.

Variational method: Calculating radial distribution function and ground state for helium

For the purpose and scope of this report, we can analyze the helium atom by first considering a single-electron helium ion (He*) and then extending the analysis to include both electrons. We will use the variational method to provide an approximate solution for the ground state energy of the helium atom.

For a single-electron helium ion (He*), the wave function in the position basis can be written as:

$$\psi_{100}^{Z}(\vec{r}) = e^{-Zr/a}\sqrt{\frac{Z^3}{\pi a^3}} \tag{2.24}$$

Here, Z is the number of protons in the nucleus (Z=2 for helium), a is the Bohr radius, and the last term is the normalization constant.

To consider the full helium atom with two electrons, we need to describe the system using a wave function $\psi(r_1, r_2)$, where r_1 and r_2 are the positions of the two electrons. The Hamiltonian operator for this system, ignoring spin and considering the zeroth-order approximation, is given by:

$$H = H_1 + H_2 + V_{ee} \tag{2.25}$$

Where H_1 and H_2 represent the Hamiltonians for each electron, and V_{ee} is the electron-electron interaction potential. The Hamiltonian can be written explicitly as:

$$H = \frac{P_1^2}{2m} + V_{coul}^{Z=2}(\vec{r}_1) + \frac{P_2^2}{2m} + V_{coul}^{Z=2}(\vec{r}_2) + V_{ee}(r_1, r_2) \tag{2.26}$$

The Coulomb potential is given by:

$$V_{coul}^{Z=2}(\vec{r}) = \frac{-Ze^2}{4\pi\epsilon_0}\frac{1}{|\vec{r}|} \tag{2.27}$$

And the electron-electron interaction potential is given by:

$$V_{ee}(r_1, r_2) = \frac{e^2}{4\pi\epsilon_0}\frac{1}{|\vec{r_1} - \vec{r_2}|} \tag{2.28}$$

To find the ground state of the helium atom, we first consider the case without electron-electron interaction. In this case, we have two separate helium ions (He*), and the system's wave function can be written as:

$$\psi(r_1, r_2) = \psi_{100}^{Z=2}(\vec{r_1})\psi_{100}^{Z=2}(\vec{r_2}) \tag{2.29}$$

The energy of this state is:

$$E = -13.6eVZ^2 * 2 \approx -109eV$$

However, this is not a good approximation since the electron-electron interaction is not a small perturbation. The energy of the electron-electron interaction is of a similar order of magnitude as the proton-electron interaction.

As there is no way to solve the problem analytically and perturbation theory is not useful in this case, we resort to the variational method using trial wave functions. Using this approach, we will employ trial wave functions to approximate the ground state energy of the helium atom. We can start with a similiar simple approximation by assuming the electrons do not interact. In this case, the trial wave function for the full Hamiltonian can be written as a product of two hydrogen-like wave functions with Z = 2, this is the same as Eq 2.29. Before proceeding, we verify that the wavefunction is normalized:

$$\int dr_1^3 dr_2^3 \left|\psi(\vec{r_1})\right|^2 = 1$$

Next, we calculate the expectation value of the Hamiltonian using the trial wave function:

$$\langle\psi| H |\psi\rangle = \langle\psi| H_1 |\psi\rangle + \langle\psi| H_2 |\psi\rangle + \langle\psi| V_{ee} |\psi\rangle$$

The first two terms in the expression correspond to the hydrogen-like helium ion (He*) energies, which can be easily computed:

$$H_1 |\psi\rangle = -13.6eV Z^2 |\psi\rangle$$
$$H_2 |\psi\rangle = -13.6eV Z^2 |\psi\rangle$$

The third term in the expression, $\langle\psi| V_{ee} |\psi\rangle$, accounts for the electron-electron interaction and requires integrating the product of the trial wave function and the electron-electron interaction potential:

$$\langle\psi| V_{ee} |\psi\rangle = \int dr_1^3 dr_2^3 \left|\psi_{100}^Z(\vec{r}_1)\right|^2 \left|\psi_{100}^Z(\vec{r}_2)\right|^2 * \frac{e^2}{4\pi\epsilon_0} |\vec{r}_1 - \vec{r}_2|$$

After evaluating the integrals, we obtain an approximate energy for the ground state of the helium atom using the trial wave function:

$$\langle\psi| H |\psi\rangle \approx -13.6eV4 - 13.6eV4 + \langle\psi| V_{ee} |\psi\rangle \approx -75eV$$

This result is an approximation for the ground state energy of helium using the trial wave function 2.29, which assumes no electron-electron interaction. Although this is a simple approximation, it provides a starting point for understanding the behavior of the helium atom. When incorporating the second electron in the helium atom, we must consider the effective charge experienced by the electrons due to the remaining helium nucleus and the first electron. If the second electron is very far away, it will perceive a net charge of $+1$, as the positive charge from the proton is neutralized by the negative charge of the first electron, which forms a symmetric wave function around the nucleus. On the other hand, if the second electron is very close to the nucleus, it will experience a charge of $+2$, as the first electron can be treated as a spherically symmetric charge distribution. This idea can be visualized by the following figure 2.3:

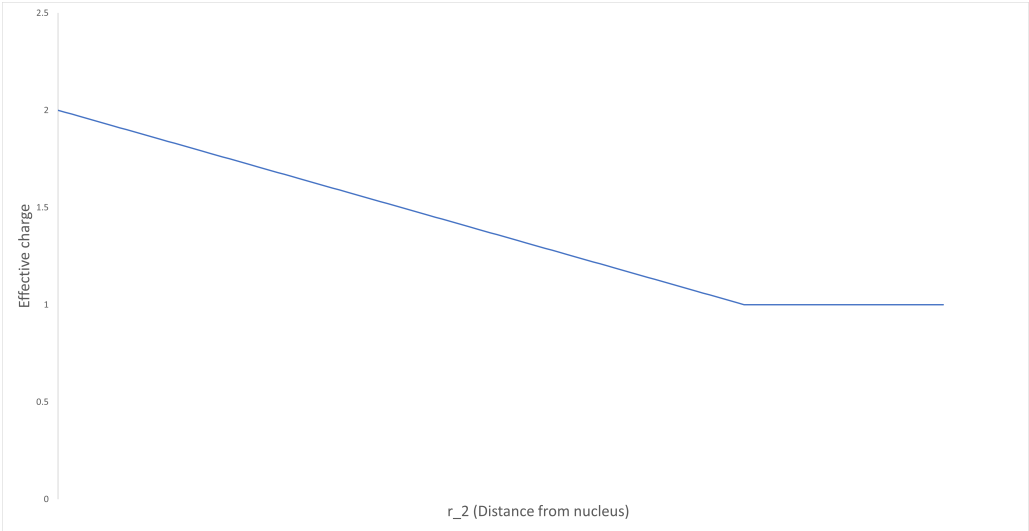


Figure 2.3: Effective charge seen by e_2^-

To account for this varying effective charge, we can introduce a new parameter, Z, in our trial wave function Eq. 2.29. Now, instead of directly using Z=2 in our expectation value calculation, we can treat Z as an arbitrary parameter and minimize the energy with respect to Z:

$$E_0 <= \langle\psi_Z|\, H_1^Z + H_1^Z + V_{ee}\,|\psi_Z\rangle$$

The expectation value can be computed as:

$$\langle\psi_Z|\, H\,|\psi_Z\rangle = \langle\psi_Z|\, H_1\,|\psi_Z\rangle + \langle\psi_Z|\, H_2\,|\psi_Z\rangle + \langle\psi_Z|\,(H_1^{Z=2} - H_1^Z) + (H_2^{Z=2} - H_2^Z) + V_{ee}\,|\psi_Z\rangle$$

By considering the difference between the Hamiltonians H_1 and H_2 with Z and Z=2, the momentum terms will cancel out, and we will be left with a function that depends only on the position, r:

$$\langle\psi_Z|\, H\,|\psi_Z\rangle = -13.6eVZ^2 - 13.6eVZ^2 + \langle\psi_Z|\, -\frac{Ze^2}{4\pi\epsilon_0}\frac{1}{|\vec{r}_1|} - \frac{Ze^2}{4\pi\epsilon_0}\frac{1}{|\vec{r}_2|} + ...\,|\psi_Z\rangle$$

The last term in the expression represents the integral of the product of the trial wave function and the electron-electron interaction potential, denoted by the ellipsis (...). After evaluating the integrals, the energy expression becomes:

$$\langle \psi_Z | \, H \, | \psi_Z \rangle = 13.6eV(2Z^2 - \frac{27}{4}Z)$$

To find the optimal value of Z, we can take the gradient of the energy expression with respect to Z and set it equal to zero (A plot of this energy expression can be seen in 2.4. This results in a minimized effective nuclear charge of Z = 1.69. Plugging this value back into the energy expression, we obtain an approximate ground state energy of:

$$E_0 <= \langle \psi_Z | \, H \, | \psi_Z \rangle = -77.5eV$$

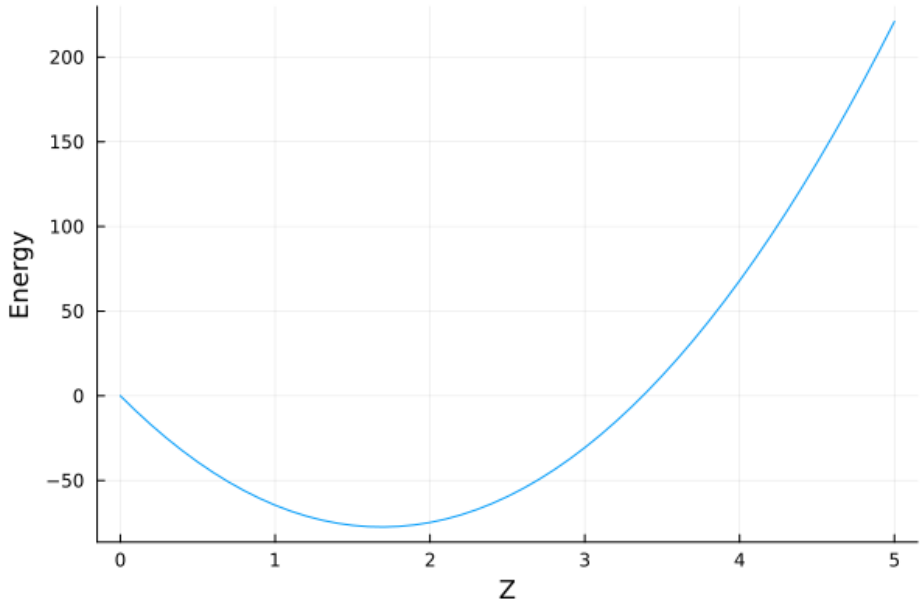


Figure 2.4: Variational energies as a function of the effective nuclear charge Z (code can be found in A.4)

This result provides an improved approximation for the ground state energy of the helium atom using the variational method with the trial wave function $\psi(r_1, r_2) = \psi^Z_{100}(r_1)\psi^Z_{100}(r_2)$ and the parameter Z as an adjustable parameter that accounts for the varying effective charge experienced by the electrons. For comparison, the experimental value for the ground state for Helium is $E_0 = -79.005eV$ [8]

Methods and Implementation

3.1 Alternating Direction Implicit Method for Two-Electron Schrödinger Equation on a 2D Grid

In this section, we will discuss the application of the Alternating Direction Implicit (ADI) method to a two-electron system in a 2D grid. The ADI method is a powerful numerical technique used to solve partial differential equations arising from the time-dependent Schrödinger equation. We will outline the algorithm and discuss its application to a two-electron system, highlighting the steps involved and the observables of interest.

The unitary, split-step finite difference approximation for the two-dimensional Schrödinger equation enables separate handling of potential and kinetic energy operators. In order to solve the ADI system, we begin by solving for the intermediate wave function, $\psi_{i,j}$, on a two-dimensional grid with unknowns in both the x and y directions. To achieve this, we fix $j = (2, 3, ..., N_y - 1)$ and solve the resulting tridiagonal system to obtain solutions for $\psi_{i,j} = (\psi^*_{2,j}, \psi^*_{3,j}, ..., \psi^*_{N_x-1,j})$. This procedure is then repeated for every interior j point.

The resulting matrices A and B can be written as:

$$\mathrm{A} = \begin{pmatrix} \kappa & \delta & & & \\ \delta & \kappa & \delta & & \\ & \delta & \ddots & \ddots & \\ & & \ddots & \kappa & \delta \\ & & & \delta & \kappa \end{pmatrix}, \mathrm{B} = \begin{pmatrix} \tau & -\delta & & & \\ -\delta & \tau & -\delta & & \\ & -\delta & \ddots & \ddots & \\ & & \ddots & \tau & -\delta \\ & & & -\delta & \tau \end{pmatrix}$$

where $\delta = -i\lambda$, $\kappa = 1 + \frac{2i}{\lambda}$, and $\tau = 1 - \frac{2i}{\lambda}$.

In the next step, we propagate the solution from $n+\frac{1}{2} \to n+1$ by solving the second equation in the ADI system. This time, we fix $i = (2, 3, ..., N_x - 1)$ to avoid any directional bias and obtain the solutions of our tridiagonal system.

The algorithm for computing the updated wave function, $\psi_{i,j}^{n+1}$, is slightly different and involves an extra step. The Julia code implementation is shown below:

```
# Define necessary variables and matrices
d = exp.(im*dt.*V/2)
s = exp.(-im*dt.*V/2)

# Solve for \psi^{n+1}(i,j) fixing i
# and solving along the y-axis
for i in 2:Nx-1
    a = \psi[i, :]
    psi[i, :] = (B / A) * a
    psi[i, :] = (psi[i, :]) .* (s[ i , : ] ./ d[i, :])
end

# Insert boundary conditions
psi[ 1 , : ] = psi[ end , : ] = psi[ : , 1 ] = psi[ : , end ] .= 0
```

Here, we've solved the second equation for each i index in two steps. First, we solve for $\psi_{i,j}^{n+1}$ by considering only the kinetic part of the operator, and then using the solution from that, we calculate the effect the potential operators have.

The two-electron system in the 2D grid is described by the time-dependent Schrödinger equation with the Hamiltonian operator $H = T + V$, where T represents the kinetic energy operator and V the potential energy operator. In the context of the two-electron system, the potential energy operator includes electron-electron interaction, as well as any external potentials acting on the system.

To model the two-electron system, we define the wave function $\psi(\mathbf{r}_1, \mathbf{r}_2, t)$, which depends on the positions of the two electrons, $\mathbf{r}_1$ and $\mathbf{r}_2$, as well as time t. The ADI method is particularly useful for systems like this, where the potential energy operator is non-separable, and the time evolution of the wave function involves both kinetic and potential energy operators.

The time evolution of the two-electron system is governed by the following steps:

1. Initialize the two-electron wave function $\psi(\mathbf{r}_1, \mathbf{r}_2, t)$ at time $t = 0$, using an appropriate initial condition.

2. Apply the ADI method to propagate the wave function in time, solving the tridiagonal systems along the r_1 and r_2 directions iteratively, as described earlier.

3. Calculate observables of interest, such as the electron density $\rho(\mathbf{r}, t) = \int |\psi(\mathbf{r}, \mathbf{r}', t)|^2 d\mathbf{r}'$, where the integral is performed over the coordinates of the second electron, $\mathbf{r}'$.

4. Continue the time propagation and calculate observables until a desired time is reached or a steady-state solution is obtained.

Using the ADI method for solving the two-electron Schrödinger equation on a 2D grid, we can study various properties of the system, such as the time evolution of the electron density, the energy spectrum, and the response of the system to external perturbations. Furthermore, the method can be extended to study more complex systems with additional particles, dimensions, or interactions.

3.2 Visscherr Method

Visscher's method [11], first proposed in 1991 can be used for solving the two-particle Schrödinger equation as it offers a sophisticated approach that builds upon the foundation of explicit methods, resulting in enhanced probability conservation and overall performance. This technique is particularly important when examining quantum systems involving multiple particles, as it can capture complex interactions and phenomena more accurately.

In the explicit and implicit methods previously discussed, the real and imaginary components of the wave function are obtained simultaneously. However, the staggered explicit method treats these components separately at staggered moments, leading to better probability conservation. To achieve this, the wave function is rewritten as

$$\psi(r_1, r_2, t) = \psi_R(r_1, r_2, t) + i\psi_I(r_1, r_2, t) \tag{3.1}$$

where R and I denote the real and imaginary components, respectively. The Schrödinger equation is then rewritten as two coupled partial differential equations, with the algorithm separating into a pair of equations for the real and imaginary components. In Visscher's method, these components are evaluated at slightly different, staggered times. This approach leverages the extra degree of freedom offered by the complex nature of the wave function to better preserve probability, which is especially important for scattering problems. From the explicit solution, we can show this algorithm separating into the two coupled equations:

$$\psi_{l,m}^{n+1} = u_{l,m}^{n+1} + iv_{l,m}^{n+1} \tag{3.2}$$

$$\begin{aligned}
u_{l,m}^{n+1} &= u_{l,m}^{n-1} + 2\left[\left\{\left(\frac{1}{m_1}+\frac{1}{m_2}\right)4\lambda + \Delta t V_{l,m}\right\} v_{l,m}^{n}\right. \\
&\quad \left. - \lambda\left\{\frac{1}{m_1}(v_{l+1,m}^{n} + v_{l-1,m}^{n}) + \frac{1}{m_2}(v_{l,m+1}^{n} + v_{l,m-1}^{n})\right\}\right] \\
v_{l,m}^{n+1} &= v_{l,m}^{n-1} - 2\left[\left\{\left(\frac{1}{m_1}+\frac{1}{m_2}\right)4\lambda + \Delta t V_{l,m}\right\} u_{l,m}^{n}\right. \\
&\quad \left. - \lambda\left\{\frac{1}{m_1}(u_{l+1,m}^{n} + u_{l-1,m}^{n}) + \frac{1}{m_2}(u_{l,m+1}^{n} + u_{l,m-1}^{n})\right\}\right]
\end{aligned} \tag{3.3}$$

As mentioned, this method solves for the two coupled equations at slightly staggered times, represented by the following equation:

$$[u_{l,m}^{n}, v_{l,m}^{n}] = [Re\psi(x,t), Im\psi[x, t+\frac{1}{2}\Delta t] \tag{3.4}$$

These specific definitions for probability density ensure that the solutions obtained from Visscher's method exhibit enhanced accuracy and stability, particularly in terms of probability conservation. By carefully considering the real and imaginary components of the wave function at staggered times, the method allows for a more precise and reliable representation of the underlying physical system.

3.3 Constructing the Potentials

3.3.1 1D Model

For the 1D scattering between the two particles, there are two potential types which can used to define what interaction the wave packets will have. By default, the Gaussian potential is chosen for its mathematical simplicity and smoothness. The code also contains a commented-out section for a square potential, which can provide an abrupt interaction between particles, leading to rapid changes in the wave packets. These two types of potentials can be used to study various interaction scenarios between the particles.

Gaussian Potential

The Gaussian potential is another widely used potential model in quantum mechanics, particularly in particle-particle collision experiments. The Gaussian potential is characterized by a smooth and continuous function that decreases exponentially as the position moves away from the center of the potential.Mathematically, the Gaussian potential can be described as follows:

$$V(x) = -V_{\max} \cdot \exp\left(-\frac{(x - x_c)^2}{2\sigma^2}\right) \tag{3.5}$$

The parameter σ determines the spatial extent of the potential and plays a crucial role in shaping the Gaussian function. A larger σ results in a wider and more spread-out potential, whereas a smaller σ leads to a narrower and more localized potential. In the provided code, the Gaussian potential is implemented within the nested loops iterating over the grid points (i, j) of the two-dimensional potential matrix v. The calculation of the Gaussian potential can be achieved with the following code snippet:

```
tmp = dx^2 / alpha^2 / 2.0
for i in 0:(N1 + 1)
    for j in 0:(N2 + 1)
        v[i + 1, j + 1] = -vmax * exp(-tmp * (i - j)^2)
    end
end
```

This implementation calculates the potential value $v[i+1, j+1]$ for each grid point (i, j) using the Gaussian potential formula. The variables dx and $alpha$ are used to scale the potential function, and $vmax$ determines the maximum depth of the potential well.

The Gaussian potential allows us to investigate various interaction scenarios in a smooth and continuous manner. Its unique properties make it suitable for studying the behavior of particles under different potential depths, widths, and shapes, as well as their response to external forces or perturbations.

Square well Potential

The square potential, also known as the square-well potential, represents a potential energy function that is constant within a certain range and zero outside of that range. It is essentially an idealized model that provides a simple way to study the behavior of particles under abrupt interactions. Mathematically, the square potential can be defined as follows:

$$V(x) = \begin{cases} V_0, & \text{if } |r - r_c| \leq \frac{\alpha}{2}\ 0, \\ \text{otherwise} \end{cases} \tag{3.6}$$

To implement the square potential in a numerical simulation, we can replace the Gaussian potential calculation in the provided code with the following code snippet:

```
if abs(i-j)*dx <= alpha
    v[i + 1, j + 1] = vmax
else
    v[i + 1, j + 1] = 0.0
end
```

This implementation sets the potential value $v[i + 1, j + 1]$ to $vmax$ if the condition $abs(i - j) * dx <= \alpha$ is met, which means the position is within the potential well. Otherwise, the potential value is set to 0. The variable dx represents the spatial grid discretization, and α defines the width of the square potential.

The square potential allows us to study the behavior of particles in various interaction scenarios, such as changing the potential from attractive to repulsive, adjusting the depth of the potential well to create quasi-bound states,

or altering the potential to be "softer" or more abrupt. The simplicity of the square potential, as well as its abrupt nature, makes it a useful tool for investigating rapid changes in the wave packets of interacting particles in quantum mechanics.

3.3.2 2D Model

For the 2-electron Schrödinger problem in 2D, we can create a function to construct the central atomic potential. The atomic potential function accounts for the interactions between the electrons and the nucleus, as well as the angular momentum of the electrons. The function *make_atomic_potential* constructs the potential energy matrix for the atomic system. The function takes the parameters object par as an input, which contains the discretization parameters, such as domain boundaries, step sizes, and the total number of grid points.

The atomic potential is composed of four components:

1. The Coulomb interaction between the electron and the nucleus, with charge Z: $-\frac{Z}{r}$
2. The effective centrifugal potential due to the angular momentum l: $\frac{l(l+1)}{2r^2}$

These components are calculated for each electron separately (denoted as electron 1 and electron 2). Here's the implementation of the *make_atomic_potential* function:

```
function make_atomic_potential(par::params)
    Z = 2
    l1 = 0
    l2 = 0

    V = zeros(par.N)

    for i in 1:par.N
        V[i] = -Z/par.x1[i] + l1*(l1+1)/(2*par.x1[i]^2)
                - Z/par.x2[i] + l2*(l2+1)/(2*par.x2[i]^2)
    end
```

```
    return V
end
```

In this function, the charge Z of the nucleus is set to 2 for helium, while the angular momentum quantum numbers l_1 and l_2 for both electrons are set to 0. The potential energy matrix V is initialized with zeros and has dimensions of par.N x par.N. The nested loops iterate through each grid point, and the potential is calculated for each electron, considering the Coulomb interaction and centrifugal potential components. The resulting potential matrix V is then returned.

The *make_atomic_potential* function provides a way to construct the atomic potential for a 2-electron system in a 2D space, allowing for the study of electron-nucleus interactions and the behavior of electrons under various potential configurations.

3.4 Radial Probability Distribution

The radial Schrödinger equation is a reformulation of the Schrödinger equation for central potentials, which depend only on the radial distance from the origin. Central potentials are spherically symmetric and do not depend on the angular coordinates (θ and ϕ). When dealing with such potentials, it is convenient to use spherical polar coordinates (r, θ, ϕ) instead of Cartesian coordinates (x, y, z).

By separating the Schrödinger equation into radial and angular parts, we can simplify the problem into two separate equations: one for the radial part and another for the angular part. The radial Schrödinger equation then describes the radial dependence of the wave function.
The radial Schrödinger equation is given by:

$$-\frac{\hbar^2}{2m}\frac{1}{r^2}\frac{d}{dr} = r^2\frac{dR(r)}{dr} + V(r) + -\frac{\hbar^2}{2m}\frac{l(l+1)}{r^2}R(r) = ER(r) \tag{3.7}$$

Here, $R(r)$ is the radial part of the wave function, $V(r)$ is the central potential, $\hbar$ is the reduced Planck constant, m is the mass of the particle, E is the energy, and l is the angular momentum quantum number. The term $\frac{l(l+1)}{r^2}$ represents the centrifugal potential that arises due to the conservation

of angular momentum when a particle is subjected to a central force, as constructed in the last section.

The radial probability distribution for a helium atom provides insight into the distribution of electrons around the nucleus as a function of the radial distance from the nucleus. Helium, with two electrons and an atomic number of 2, has an electron configuration of $1s^2$. This means that both of its electrons reside in the 1s orbital.

The 1s orbital has the quantum numbers n = 1, l = 0, and m = 0, where n is the principal quantum number, l is the angular momentum quantum number, and m is the magnetic quantum number. The radial wave function for the 1s orbital of a hydrogen-like atom (including helium) can be expressed as:

$$R(r) = \frac{1}{\sqrt{\pi a_0^3}} exp(-r/a_0) \tag{3.8}$$

where a is the Bohr radius, and r is the radial distance from the nucleus. The radial probability distribution is the square of the absolute value of the radial wave function multiplied by the surface area of a spherical shell at radius r:

$$P(r) = |R(r)|^2 \, 4\pi r^2 \tag{3.9}$$

For the helium atom's 1s orbital:

$$P(r) = (4/a_0^3) exp(-2r/a_0) r^2 \tag{3.10}$$

This function represents the probability of finding an electron in a small volume at a radial distance r from the nucleus. The radial probability distribution for helium's 1s orbital has a maximum value at $r = a_0/2$, indicating that the electrons in the 1s orbital are most likely to be found at half the Bohr radius from the nucleus. This probability distribution provides a more accurate representation of the electron distribution in the helium atom compared to the Bohr model, as it takes into account the wave nature of the electrons.

We can make a rough plot of the radial probability distribution for a two-electron system, such as helium, using Julia. To achieve this, we first simplify the model by neglecting the electron-electron interaction, allowing us to treat the wave function of the helium atom as a product of two hydrogen-like wave

functions. This approximation is not highly accurate but provides a basic understanding of the helium atom's radial probability distribution.

We begin by importing the necessary Julia packages: QuadGK for numerical integration, and Plots for plotting the results. Next, we define the constants for the problem: the nuclear charge Z is set to 2 for helium, the Bohr radius a0 is defined as 0.5291772108 Å, and R_max is set to 10, defining the maximum range for the radial distance.

The wave function for the two-electron system is defined using the psi function, which calculates the product of two hydrogen-like wave functions (one for each electron) with radial parts given by R_1 and R_2. The radial wave function for each electron is represented by $R_1 = \frac{1}{\sqrt{\pi}}(Z/a0)^{3/2}exp(-(Zr_1)/a_0)$ and $R_2 = \frac{1}{\sqrt{\pi}}(Z/a_0)^{3/2}exp(-(Zr_2)/a_0)$.

To calculate the probability density, we create the P function that squares the absolute value of the wave function $\psi(r1, r2)$. The integrand function integrand is defined to calculate the probability density multiplied by the square of the radial distance r_1, i.e., $P(r1, r2)r_1^2$. This function will be used for numerical integration.

The calculate_P_r2 function integrates the integrand function over the radial distance r_1 from 0 to R_max using the quadgk function for numerical integration. This allows us to calculate $P(r_2)$ for a range of r_2 values.

We can create an array of r_2 values from 0 to 2 with a length of 100 and calculate $P(r_2)$ values for each r_2 value in the array using the calculate_P_r2 function. The calculated $P(r_2)$ values are then plotted against the r_2 values. This gives the following distribution in figure 3.1

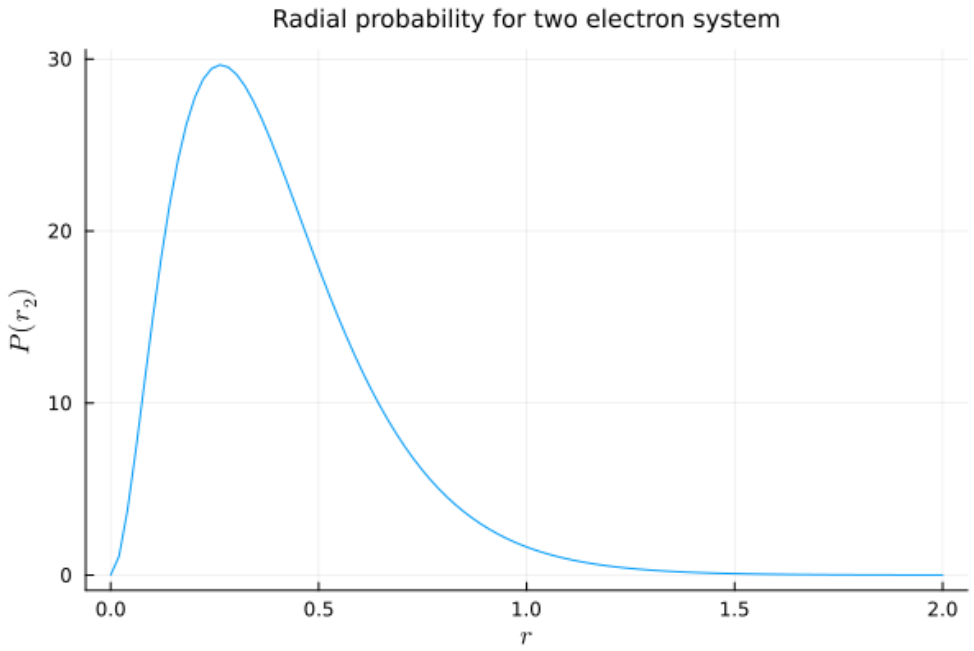


Figure 3.1: Approximate Radial Probability Distribution for two electron

It is important to note that this code provides only a rough approximation of the radial probability distribution for a helium atom by ignoring the electron-electron interaction and treating the wave function as a product of two hydrogen-like wave functions. Although this approximation can offer a basic understanding of the helium atom's radial probability distribution, it may not be accurate enough for more detailed analysis.

Results

4.1 1D

4.1.1 Initial Wave function and boundary conditions

In our study, we analyze a scattering experiment involving two particles. Particle 1 is initially located at position r_1^0 with momentum k_1, while Particle 2 is initially far away at position r_2^0 with momentum k_2. To model this scenario, we assume that the wave functions of these particles are represented by independent wave packets, as given by:

$$\psi(r_1, r_2, t = 0) = e^{ik_1 r_1} exp[-\frac{(r_1 - r_1^0)^2}{4\sigma^2}] e^{ik_2 r_2} exp[-\frac{(r_2 - r_2^0)^2)}{4\sigma^2}] \tag{4.1}$$

Each term in the product consists of a plane wave part ($e^{(ik_1 r_1)}$ for particle 1 and ($e^{(ik_1 r_2)}$ for particle 2) and a Gaussian factor $exp[-\frac{(r_1 - r_1^0)^2}{4\sigma^2}]$ for particle 1 and $exp[-\frac{(r_2 - r_2^0)^2)}{4\sigma^2}]$ for particle 2). The plane wave part represents the momentum of the particles, while the Gaussian factor localizes the particles around their initial positions (r_1^0 and r_2^0).

The presence of Gaussian factors means that the state of the system is not an eigenstate of the momentum operators ($-i\frac{\partial}{\partial r_j}$) for particles 1 and 2. In other words, the Gaussian factors introduce a spread of momenta around the mean values k_1 and k_2. This is because the Gaussian factors describe the spatial distribution of the particles, which, in turn, affects the momenta of the particles. A well-localized particle (a small σ) will have a broader range of momenta, while a delocalized particle (a large σ) will have a narrower range of momenta.

When the width of the Gaussian wave packet (σ) becomes very large, approaching infinity ($\sigma \to \infty$), the particles become increasingly delocalized. As a result, the spatial distribution of the particles becomes uniform, and the Gaussian factors no longer have a significant effect on the momenta of the particles. In this case, the state of the system approaches a momentum

eigenstate, meaning that the particles' momenta are well-defined and essentially determined by the plane wave parts of the wave function.

Further investigation into our initial wave function shows that in a two-particle system, such as the one described in the scattering experiment, the Schrödinger equation can be simplified by transforming it into a system of two coupled equations. This transformation is achieved by expressing the system in terms of the relative coordinate (x) and the center-of-mass coordinate (X). The relative coordinate (x) represents the difference in positions of the two particles, while the center-of-mass coordinate (X) represents the average of their positions, weighted by their masses. This transformation can be defined as follows:

$$\begin{aligned} x &= x_1 - x_2 \\ X &= \frac{(m_1 x_1 + m_2 x_2)}{(m_1 + m_2)} \end{aligned} \tag{4.2}$$

By transforming the Schrödinger equation into this new coordinate system, we can separate it into two equations, one for the relative coordinate (x) and another for the center-of-mass coordinate (X). This separation simplifies the analysis of the system, as each equation can be solved independently.

However, the initial condition given by equation 4.1 cannot be separated into distinct conditions for x and X. This is because the wave function in equation 4.1 is given as a product of independent wave packets for particles 1 and 2, and the Gaussian factors introduce dependencies between the particles' positions. As a result, we cannot express the initial condition as a simple product of functions of x and X.

Due to this limitation, a solution for each particle's coordinate is required, instead of just solving for the relative coordinate (x) and center-of-mass coordinate (X). This means that the problem cannot be reduced to a single-particle problem, and we need to solve the full two-particle Schrödinger equation.

As discussed in the last section, we can employ Visscher's method, which is a staggered-time algorithm to address the time evolution of the wave function in the scattering experiment. This approach commences with the real component of the wave function 4.1 at the initial time t = 0 and proceeds with the imaginary component at t = $\Delta t/2$. By making the assumptions

that $\Delta t/2$ is sufficiently small and the Gaussian wave packet width (σ) is adequately large, we ensure that the initial time dependence of the wave packet aligns with the plane wave components:

$$Im\psi(r_1, r_2, t = \frac{\Delta t}{2}) = sin\Bigg[k_1 r_1 + k_2 r_2 - \left(\frac{k_1^2}{2m_1} + \frac{k_2^2}{2m_2}\right)\frac{\Delta t}{2}\Bigg] \times exp\Bigg[-\frac{(r_1 - r_1^0)^2 + (r_2 - r_2^0)^2}{2\sigma^2}\Bigg] \tag{4.3}$$

For our model, a particle or projectile is introduced from an infinite distance and the scattered particles are detected at infinity as well. To model this scenario, we solve our partial differential equation within a finite domain, represented by a box with sides of length L. It is crucial that the dimensions of this box are significantly larger than the range of the interaction potential and the width of the initial wave packet to accurately represent the infinite spatial extents of the experiment.

By confining the problem to a finite domain, we impose specific boundary conditions on the wave function. These boundary conditions ensure that the wave function vanishes at the edges of the box, effectively simulating the behavior of the particles as they enter and leave the scattering region at infinity:

$$\psi(0, r_2, t) = \psi(r_1, 0, t) = \psi(L, r_2, t) = \psi(r_1, L, t) = 0 \tag{4.4}$$

These boundary conditions, along with the partial differential equation, form the foundation for our computational model, enabling us to approximate the behavior of the particles in the scattering experiment.

In table 1, it displays the parameters for asymmetrical identical particles (equal mass) to represent two particles scattering in 1D.

Table 4.1: Initial values of code parameters

Pararameter	**Value**	**Meaning**
Δr	0.01	space step
Δt	$0.5dx^2$	time step
k_1	200	Particle 1 wave vector (momentum)
k_2	-200	Particle 2 wave vector (momentum)
σ	0.05	Width of each wave packet
r_1^0	0.25	Initial value of particle 1
r_2^0	0.75	Initial value of particle 2
$N_1 = N_2$	199	Number of space steps inside
L	201	Size of box
T	20000 time steps	Total duration of model
V_0	-50000	Size of Potential depth
α	0.05	Size of Potential Width

4.1.2 Examining the Effects of Symmetry, Identical Particles, and Different Potentials

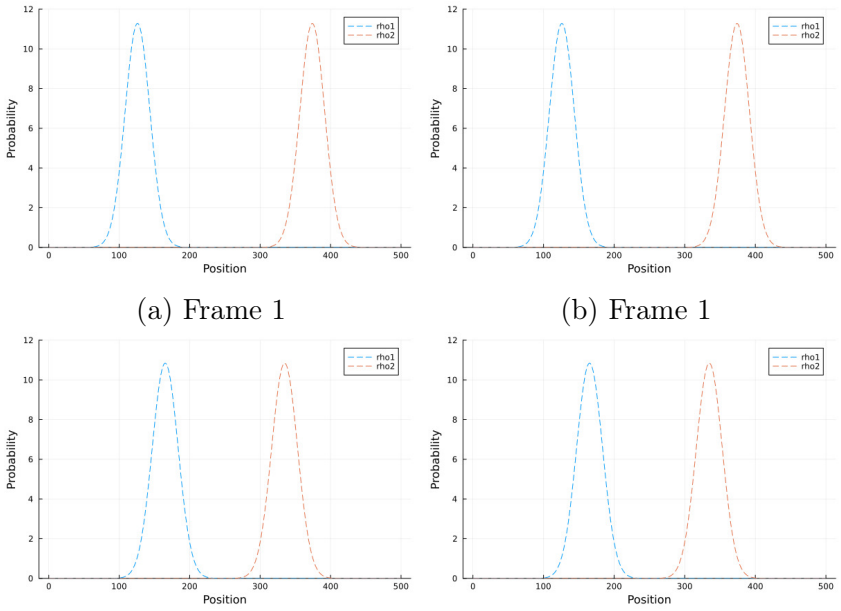
For this section of the report, we aim to investigate the behavior of particles under various conditions, including the effects of symmetry and identical particles, as well as the use of Gaussian and square potentials (constructed in Methods and Implementations). Our problem is solved in the center-of-momentum system, considering two particles with the constraint $k_2 = -k_1$.

Our findings include several scattering events, where animations can provide valuable insights. In one case, we show frames from an animation of the two-particle density as a simultaneous function of particle positions. These

densities demonstrate the penetration of packets, repulsive interactions, and the behavior of wavepackets as they move, reflect, and get trapped.
It is worth noting that there are numerical artifacts visible at early times in the simulations. Small wavepackets move in opposite directions to the larger wavepackets for each particle, which should be eliminated by the initial conditions. However, they persist due to the nature of the solutions of the Schrödinger equation.

ρ_1 and ρ_2 vs time for Gaussian potential

We can examine the various affects that the two particles will have from Gaussian potential, so far from the computation animations, there seems to be softer collisions which exhibit behaviors similar to classical particles bouncing off each other, with squeezing and broadening of the wavepackets but little breakup or capture. It's also notably smooth as it evolves in time. The following figures compare the attractive and repulsive Gaussian potential on the two particle system as it develops in time:



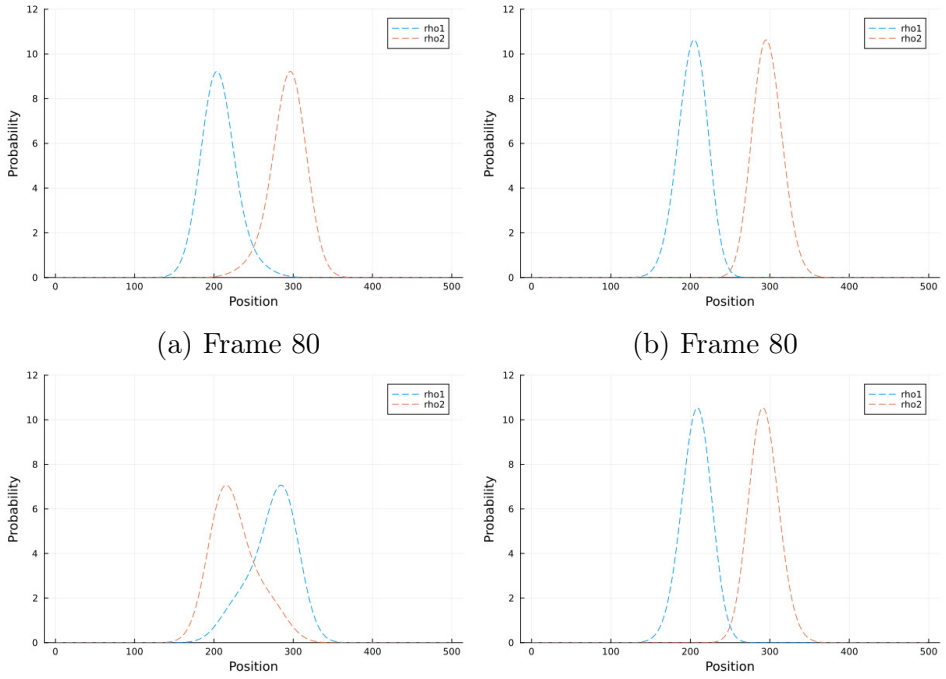
(a) Frame 1

(b) Frame 1

(c) Frame 40

(d) Frame 40

Figure 4.1: Comparison of attractive and repulsive Gaussian wave functions (Page 1)



(a) Frame 80

(b) Frame 80

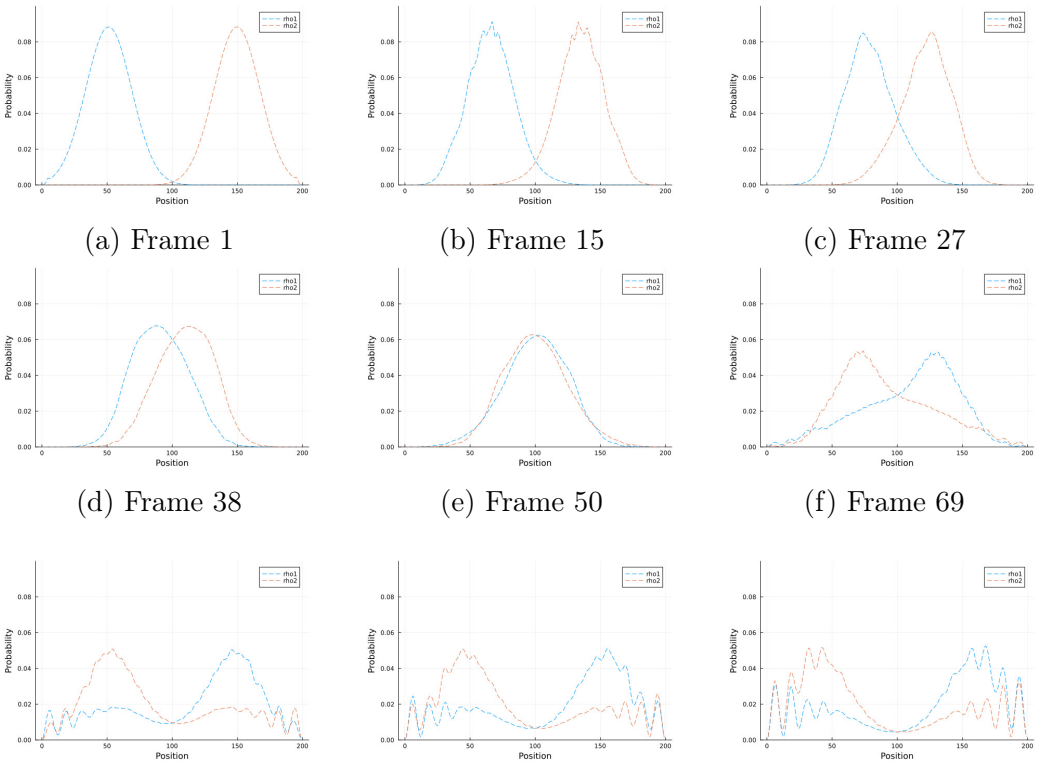
(c) Frame 120

(d) Frame 120

Figure 4.2: Comparison of attractive and repulsive Gaussian wave functions (Page 2)

ρ_1 and ρ_2 vs time for square potential

Under a square potential and depending on whether symmetry or anti symmetry (or none) is applied to the wave function, the following frames in figure 4.3 are acquired for identical anti symmetric attractive wave function. Upon examining the attractive anti symmetric wave function, we observe that the wave packets display a tendency to accelerate towards one another. As they collide, they transmit through each other, leading to an interesting dynamic following the collision. Post-collision, two new waves emerge for each wave packet, characterized as transmitted and reflected waves. Notably, these newly formed waves carry portions of the other wave packet, showcasing an intricate interplay between the colliding particles.



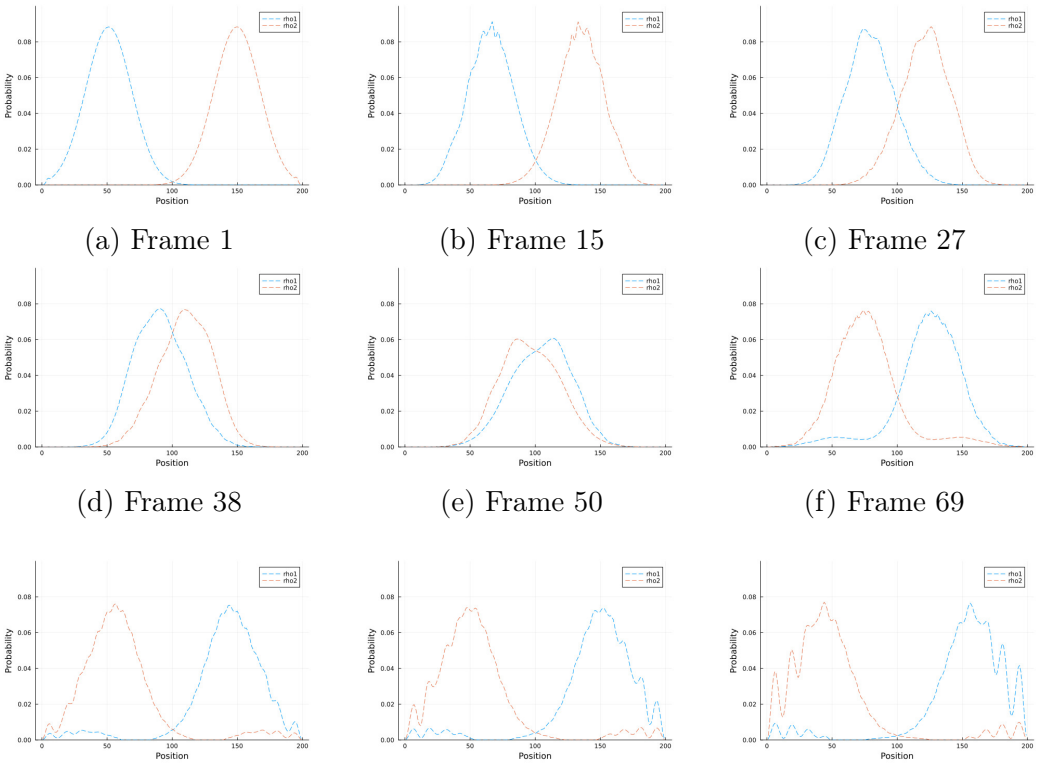
(a) Frame 1 (b) Frame 15 (c) Frame 27

(d) Frame 38 (e) Frame 50 (f) Frame 69

(g) Frame 87 (h) Frame 93 (i) Frame 99

Figure 4.3: Two particles colliding with attractive anti symmetric wave functions

When we change the sign for the square potential, the interaction between the two particles should become repulsive, this appears to be somewhat true as the two particles reflect each other instead of transmitting. This can be seen in the following figures 4.4:



(a) Frame 1 (b) Frame 15 (c) Frame 27

(d) Frame 38 (e) Frame 50 (f) Frame 69

(g) Frame 87 (h) Frame 93 (i) Frame 99

Figure 4.4: Two particles colliding with repulsive anti symmetric wave functions

As we can see from imposing anti symmetry to the wave function in the code shows effective repulsion and attraction, respectively, impacting the penetration and behavior of the wavepackets.

4.2 2D scattering

In this section, we investigate the scattering of two electrons in a two-dimensional (2D) system, taking into account atomic center potentials. We employ the Alternating Direction Implicit (ADI) method to numerically solve the two-electron Schrödinger equation using finite differences on a 2D grid. It is important to note that the electron-electron interaction is not considered in this analysis.

The time-independent Schrödinger equation for two electrons interacting with atomic center potentials, $V_1(r_1)$ and $V_2(r_2)$, in the absence of electron-electron interaction, can be written as:

$$\left[-\frac{\hbar^2}{2m_e}\left(\nabla_{r_1}^2 + \nabla_{r_2}^2\right) + V_1(r_1) + V_2(r_2)\right]\Psi(r_1, r_2) = E\Psi(r_1, r_2), \qquad (4.5)$$

where m_e is the electron mass, $\hbar$ is the reduced Planck constant, r_1 and r_2 represent the positions of the two electrons, and E is the total energy of the system.

To solve this equation numerically, we discretize the 2D spatial domain using a uniform grid with spacing Δx and Δy. The second-order central finite difference approximation is applied to approximate the Laplacian operators $\nabla_{r_1}^2$ and $\nabla_{r_2}^2$. This transforms the continuous equation into a system of linear equations that can be solved iteratively using the ADI method.

The ADI method is particularly well-suited for solving large-scale, time-dependent problems. It breaks the original problem into smaller, one-dimensional subproblems that can be solved more efficiently. In each step, the method alternates between solving the equation in the x-direction and in the y-direction, allowing for faster convergence and improved numerical stability.

To account for boundary conditions, we assume the wave function $\Psi(r_1, r_2)$ vanishes at the edges of the 2D grid, which corresponds to infinite potential barriers. This choice ensures that the electrons remain within the computational domain.

Once the wave function is obtained, we can compute the 2D scattering process. In particular, we can analyze the impact of varying the atomic center potentials, $V_1(r_1)$ and $V_2(r_2)$, on the scattering behavior, providing insights into the influence of different atomic species and their electronic configurations.

Given the initial wave function $\Psi(r_1, r_2, t = 0)$:

$$\Psi(r_1, r_2, t = 0) = e^{ik_1 r_1} \exp\left[-\frac{(r_1 - r_{01})^2}{4\sigma^2}\right] \cdot e^{ik_2 r_2} \exp\left[-\frac{(r_2 - r_{02})^2}{4\sigma^2}\right], \quad (4.6)$$

where k_1 and k_2 are the initial wave vectors of the two electrons, r_{01} and r_{02} represent the initial positions of the electrons, and σ is a parameter controlling the spatial extent of the Gaussian wave packets.

This initial wave function represents two Gaussian wave packets centered at r_{01} and r_{02}, with well-defined initial momenta given by k_1 and k_2. The product of two Gaussian wave packets implies that the electrons are initially localized around their respective positions, and their motion is independent of each other, consistent with the absence of electron-electron interaction in our analysis.

To incorporate this initial wave function into the numerical solution, we set the values of the discretized wave function on the 2D grid according to the expression for $\Psi(r_1, r_2, t = 0)$. This can be achieved by evaluating the given formula at the grid points (x_i, y_j) and (x_k, y_l), corresponding to the positions r_1 and r_2, respectively:

$$\Psi_{ijkl}(t = 0) = e^{ik_1 x_i} \exp\left[-\frac{(x_i - x_{01})^2 + (y_j - y_{01})^2}{4\sigma^2}\right] \cdot e^{ik_2 x_k} \exp\left[-\frac{(x_k - x_{02})^2 + (y_l - y_{02})^2}{4\sigma^2}\right], \quad (4.7)$$

where $\Psi_{ijkl}(t = 0)$ represents the initial value of the wave function at the grid points (x_i, y_j) and (x_k, y_l).
Once the initial wave function is set on the grid, we can proceed with solving the discretized Schrödinger equation using the Alternating Direction Implicit

method, as described in the previous response. After obtaining the time-dependent wave function, we can create animations of the two electrons.

4.2.1 Scattering off a atomic potential

For this section, we'll simulate scattering events for the two electron Gaussian wave packet with a centre atomic potential. This potential and the way it's constructed is described in section 2.3 and 3.3.2. The boundary points for the two-electron system are given by $r_1 \in [0, 30]$ and $r_2 \in [0, 30]$. The domain is discretized into a grid, and the input parameters for the Gaussian wave packets representing the two electrons are provided in table 2.

Parameter	Description	Electron 1	Electron 2
r_{0_i}	Initial position of the Gaussian wave packet	0	0
k_i	Initial wave vector	4	4
σ	Parameter controlling the spatial extent of the Gaussian wave packet	3.5	3.5
$\Delta r_1 = \Delta r_2$	Spatial discretization step	0.1	0.1
Δt	Time Step	0.005	0.005

Table 9: Input Parameters for the Gaussian Wave Packets of Two Electrons

In the two-electron system, examining different configurations indeed leads to a variety of interesting phenomena. One particularly notable observation is the effect of the angular momentum parameter in the potential on the propagation of the wave function.

As shown in Figure 4.10, the correlated wave function starts near the origin of the 2D grid and scatters as it propagates diagonally across the grid. The behavior of the wave function is characterized by interference patterns, manifesting as alternating peaks and troughs in the propagated wave function. This interference arises from the superposition of different components of the wave function, each corresponding to a different angular momentum state.

The angular momentum parameter in the potential determines how the potential interacts with the electron wave functions and affects their propaga-

tion. As the angular momentum parameter changes, the relative contributions of different angular momentum states to the overall wave function are modified, leading to a redistribution of the probability density and different interference patterns. Consequently, the scattering dynamics and the final state of the two-electron system can be significantly influenced by the angular momentum parameter in the potential.

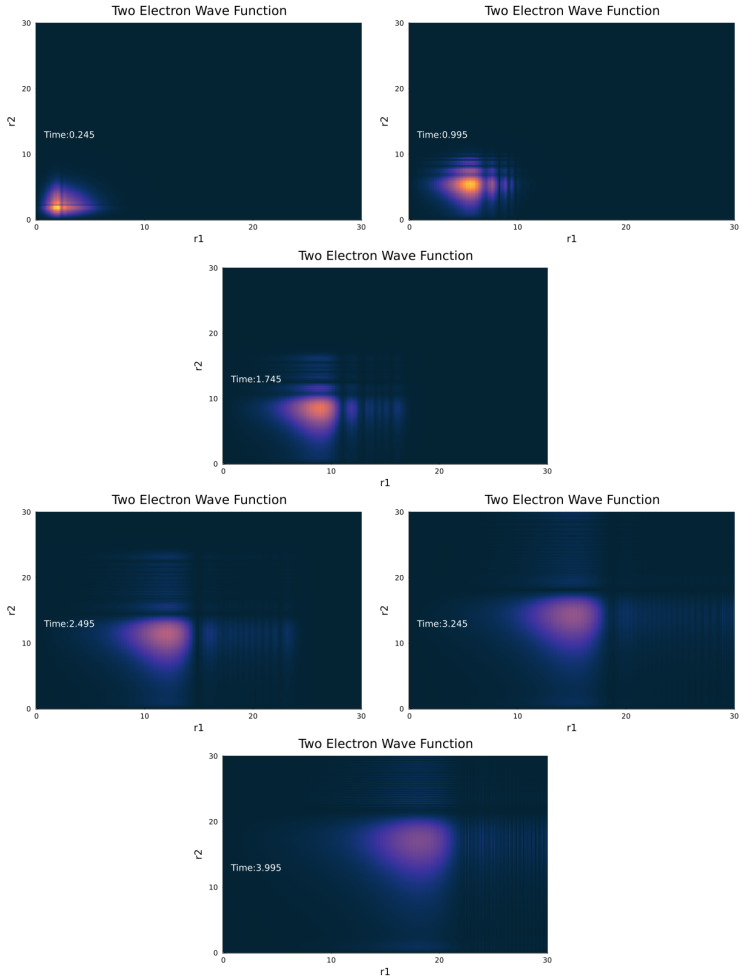


Figure 4.5: 2D scattering for 2 electron with $l_1 = 0, l_2 = 0$

Varying the value for the momentum for the two electron system gave the following figures. The angular momentum was tested between 0-4, with the

first being already plotted in 4.10.

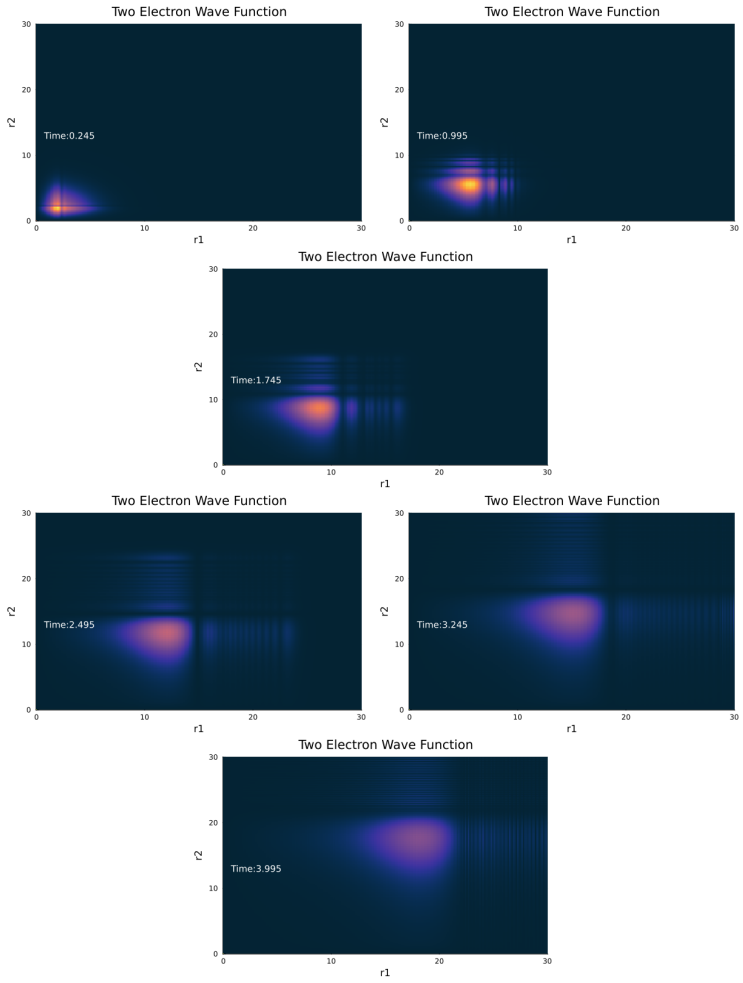


Figure 4.6: 2D scattering for 2 electron with $l_1 = 1, l_2 = 1$

While examining the heatmap representation of the 2D two-electron grid does not reveal any significant differences, a closer inspection of the wave function's final frame using a surface plot brings to light some noteworthy changes. In particular, the surface plot demonstrates a reduction in interference patterns on the grid as the system evolves.

The less pronounced interference in the surface plot may be attributed to the fact that the surface plot emphasizes the amplitude of the wave function,

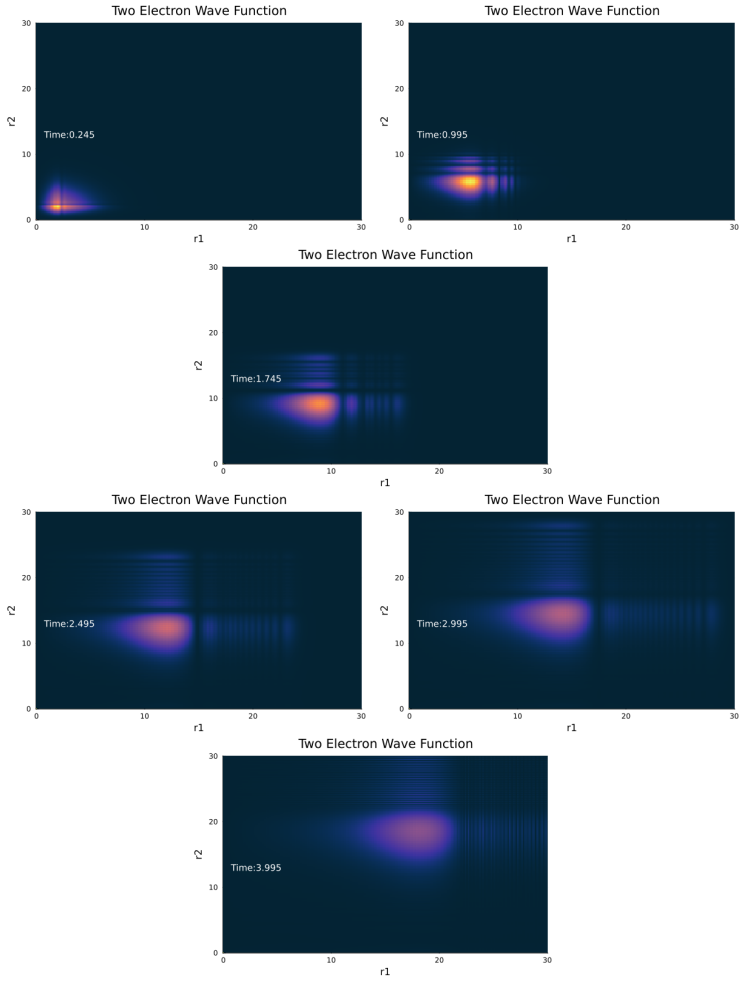


Figure 4.7: 2D scattering for 2 electron with $l_1 = 2, l_2 = 2$

making it easier to discern variations in the probability density. On the other hand, the heat map representation focuses on the intensity of the wave function, which can sometimes mask subtle changes in the interference patterns.

In conclusion, the angular momentum parameter plays a crucial role in the scattering dynamics of the two-electron system. By analyzing various configurations, it is possible to gain insights into the intricate relationship between the potential and the wave function's propagation, interference patterns, and scattering outcomes. This understanding can further inform the develop-

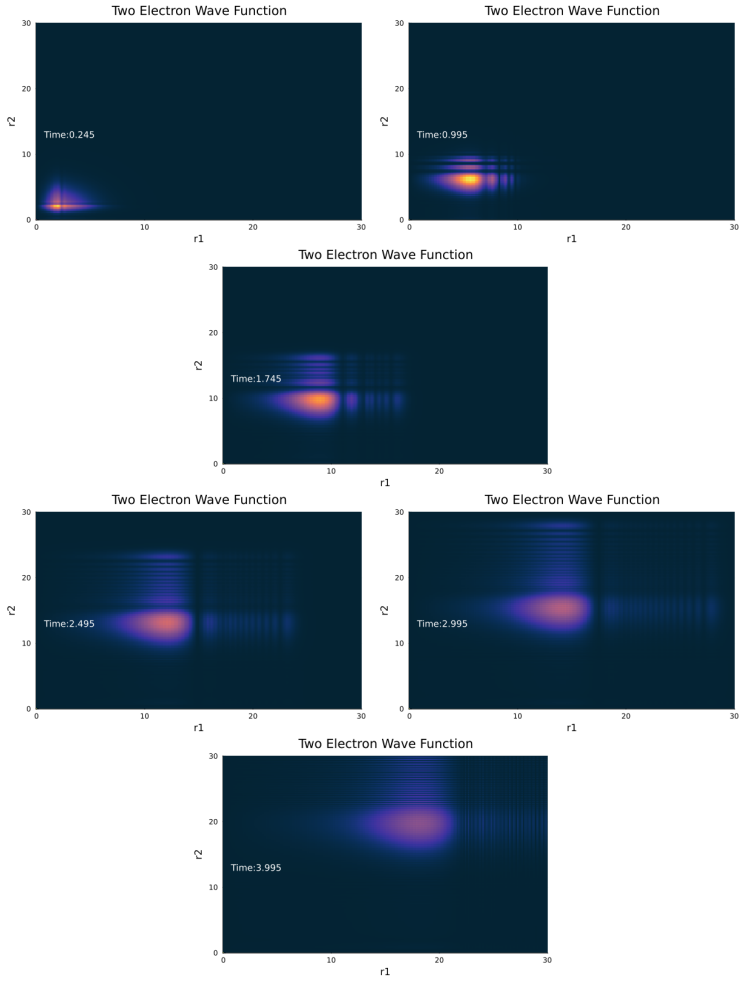


Figure 4.8: 2D scattering for 2 electron with $l_1 = 3, l_2 = 3$

ment of strategies to control and manipulate two-electron quantum systems in practical applications.

In the present work, the interaction between the two electrons was not incorporated into the code, as the inclusion of such interactions exceeded the scope of the project. However, it is important to acknowledge that the electron-electron interaction plays a crucial role in shaping the dynamics of multi-electron systems and can lead to the emergence of various phenomena such as screening, correlation, and exchange effects.

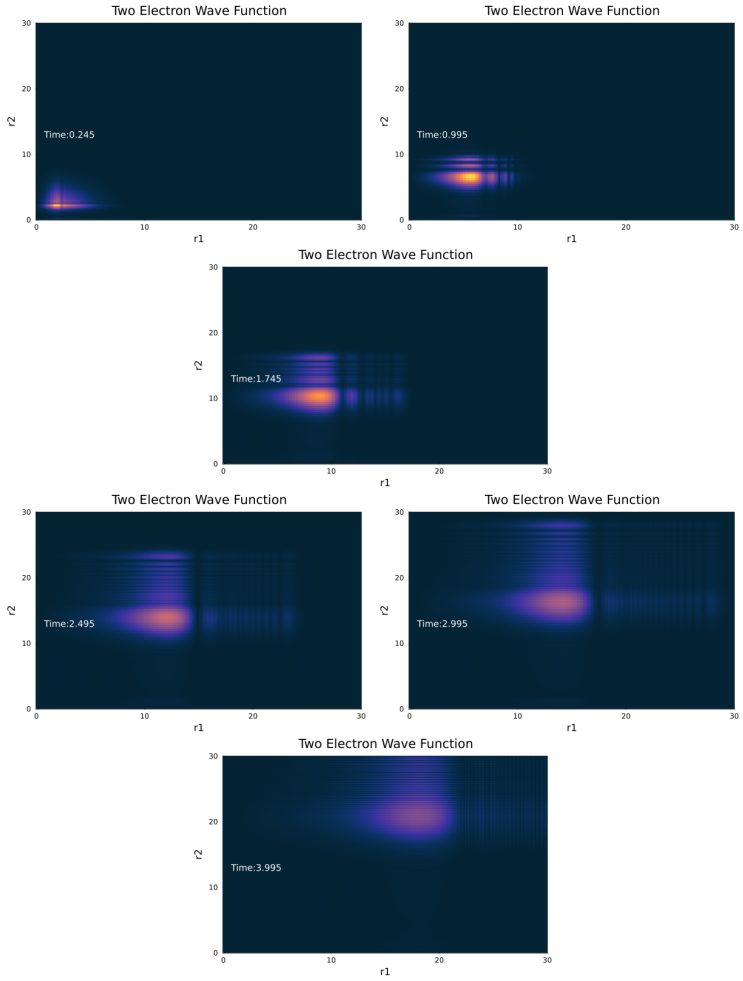


Figure 4.9: 2D scattering for 2 electron with $l_1 = 4, l_2 = 4$

In general, there are several approaches available to account for electron-electron interactions in quantum systems. One prominent method is the Configuration Interaction (CI) method, which constructs a many-body wave function by forming linear combinations of Slater determinants. This method can capture electron correlation effects by considering excitations from the ground state to higher excited states. Another approach is the Coupled-Cluster (CC) method, which models electron correlations through exponential cluster operators acting on a reference determinant.

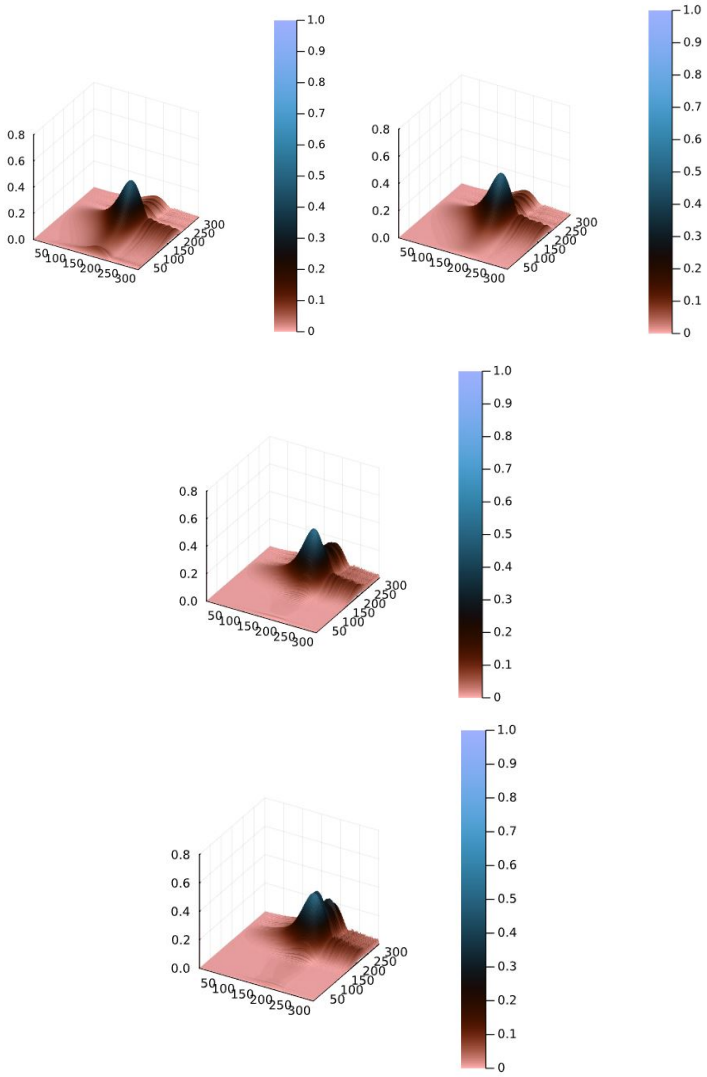


Figure 4.10: 2D scattering for 2 electron, surface plot

Other methods, such as Density Functional Theory (DFT) and Time-Dependent Density Functional Theory (TDDFT), use the electron density as the primary variable to describe the system and rely on approximations to the exchange-correlation functional to incorporate electron-electron interactions. The choice of the appropriate method depends on the specific problem at

hand, the level of accuracy required, and the available computational resources.

In future research or extensions of the current project, it would be worthwhile to investigate the effects of electron-electron interactions on the scattering dynamics of the two-electron system. Incorporating these interactions may lead to a more accurate and comprehensive understanding of the underlying physics, and could unveil new phenomena and insights into the behavior of correlated electron systems.

Summary and Conclusions

In this thesis, we have presented various methods for numerically solving the Time-Dependent Schrödinger Equation (TDSE) in the finite difference domain for one and two dimensions, focusing on the case of two-particle scattering. We have investigated the use of the Visscher method and the Alternating Direction Implicit (ADI) method for propagating the TDSE in both one-dimensional and two-dimensional systems.

A comparison in the theory section of this report for explicit and implicit methods for one-dimensional wavepacket propagation revealed the advantages and disadvantages of each approach, with particular emphasis on their stability and unitary. We managed to create a code which will solve for the two particle Schrodinger equation for its real and imaginary components at staggered times, as well we constructed different potentials and imposed different symmetry conditions onto the wave function for study.

For the radial Schrödinger equation, we solved the probability distribution and determined the ground state using variational methods through different trial wave functions. This allowed us to analyze the behavior of two particles in one-dimensional systems under various conditions.

Furthermore, we examined the two-dimensional scattering of two particles using the ADI method, which is based on the work of [6]. Although the electron-electron interaction was not included in the present study, it remains an important avenue for future research. In our two-dimensional analysis, we attempted to evaluate the meaning behind changing the various parameters for the atomic potential.

In conclusion, this thesis has provided a comprehensive overview of various numerical methods used to study the scattering of two particles in one and two-dimensional systems. The insights gained from these investigations offer valuable contributions to the field of quantum mechanics and could serve as a foundation for future research on multi-particle scattering processes. By incorporating additional factors, such as electron-electron interactions

and higher-order boundary conditions, future studies could lead to a more accurate and complete understanding of the complex behavior of correlated quantum systems.

Appendix

Listing A.1: Numerical Solution of time-dependent Schroedinger Equation for wavepacket - wavepacket scattering

```
# Global variables
global dx, alpha, vmax, N1, N2, v
global Rho, w, Ptot_i, Ptot, rho_1, rho_2, corr, SumRho
global dt, ImPsi, RePsi, con2, dtx, dm12, dm22, nprint, Nt
global out8, out10, out13

const xDim = 201

# Initialize the global variables and other parameters as required
Nt = 20000
k1 = 220.0
k2 = 220.0
dx = 0.01
dt = 0.5*dx^2
sig = 0.05
x01 = 0.25
x02 = 0.75
m1 = 1.0
m2 = 1.0
vmax = -50000.0
alpha = 0.05
nprint = 300

N1 = xDim - 2
N2 = xDim - 2
dm1 = 1 / m1
dm2 = 1 / m2
dm12 = 0.5 / m1
dm22 = 0.5 / m2
dxx2 = dx * dx
dtx = dt / dxx2
con = -1 / (2 * dxx2)
con2 = (dm1 + dm2) * dtx


w = zeros(3)

w[1] = dx/3
w[2] = 4*dx/3
w[3] = 2*dx/3

v = zeros(N1 + 2, N2 + 2)
Rho = zeros(N1 + 1, N2 + 1)

Ptot_i = 0.0
Ptot = 0.0

rho_1 = zeros(N1 + 2)
rho_2 = zeros(N1 + 2)
corr = zeros(N1 + 2)
SumRho = zeros(N1 + 2)

ImPsi = zeros(N1 + 1, N2 + 1, 2)
RePsi = zeros(N1 + 1, N2 + 1, 2)

# choice = 0, Rho
# choice = 1, rho_1
# choice = 2, corr(x)
# choice = 3, rho_1 +

choice = 1

function Initialize(out14)
```

```
    global Eri, Eii
    ww = (k1*k1/(2*m1) + k2*k2/(2*m2))*0.5*dt

    # Initialize wave function
    x1 = 0.
    for i in 1:N1
        x1 += dx
        x2 = 0.
        for j in 1:N2
            x2 += dx
            y = k1*x1 - k2*x2
            y -= ww
            a1 = (x1 - x01)/sig
            a2 = (x2 - x02)/sig
            a4 = exp(-(a1*a1 + a2*a2)/2)
            RePsi[i,j,1] = a4*cos(y)
            ImPsi[i,j,1] = a4*sin(y)
        end
    end

    # set wf to zero on boundary
    for j = 1:N2+1
        RePsi[N1+1,j,1] = 0.0
        RePsi[1,j,1] = 0.0
    end
    for i = 1:N1
        RePsi[i,N2+1,1] = 0.0
        RePsi[i,1,1] = 0.0
    end

    # Initial (unnormalized) energy  t = 0
    Eri = 0.0
    Eii = 0.0
    p = 1
    for i in 2:N1-1
        k = 1
        if p % 3 == 0
            p = 1
        end
        for j in 2:N2-1
            if k % 3 == 0
                k = 1
            end
            a1 = -2*(dm1 + dm2 + dx^2*v[i,j])*(RePsi[i,j,1]^2 + ImPsi[i,j,1]^2)
            a2 = dm1*(RePsi[i,j,1]*(RePsi[i+1,j,1] + RePsi[i-1,j,1]) + ImPsi[i,j,1]*(ImPsi[i+1,j,1]
            + ImPsi[i-1,j,1]))
            a3 = dm2*(RePsi[i,j,1]*(RePsi[i,j+1,1] + RePsi[i,j-1,1]) + ImPsi[i,j,1]*(ImPsi[i,j+1,1]
            + ImPsi[i,j-1,1]))
            ai1 = dm1*(RePsi[i,j,1]*(ImPsi[i+1,j,1] + ImPsi[i-1,j,1]) - ImPsi[i,j,1]*(RePsi[i+1,j,1]
            + RePsi[i-1,j,1]))
            ai2 = dm2*(RePsi[i,j,1]*(ImPsi[i,j+1,1] + ImPsi[i,j-1,1]) - ImPsi[i,j,1]*(RePsi[i,j+1,1]
            + RePsi[i,j-1,1]))
            Eri += w[k]*w[p]*con*(a1 + a2 + a3)
            Eii += w[k]*w[p]*con*(ai1 + ai2)
            k += 1
        end
        p += 1
    end
    # print initial Energy
    println(out14, "E (unnormalized) initial = ", Eri)
    println(out14, "E imaginary initial = ", Eii)
end

function Energy(n, out11, out12)
    # calculate the total energy of system

    Er = 0
    Ei = 0
    p = 1 # wf zero at boundaries, no p,k = 0
    for i in 2:N1
        k = 1
        if p % 3 == 0
            p = 1
        end
        for j in 2:N2
            if k % 3 == 0
                k = 1
```

```
            end
            a1 = -2*(dm1+dm2+dx*dx*v[i,j])*(RePsi[i,j,1]^2 + ImPsi[i,j,1]^2)
            a2 = dm1*(RePsi[i,j,1]*(RePsi[i+1,j,1]+RePsi[i-1,j,1]) + ImPsi[i,j,1]*(ImPsi[i+1,j,1]
            +ImPsi[i-1,j,1]))
            a3 = dm2*(RePsi[i,j,1]*(RePsi[i,j+1,1]+RePsi[i,j-1,1]) + ImPsi[i,j,1]*(ImPsi[i,j+1,1]
            +ImPsi[i,j-1,1]))
            ai1 = dm1*(RePsi[i,j,1]*(ImPsi[i+1,j,1]+ImPsi[i-1,j,1]) - ImPsi[i,j,1]*(RePsi[i+1,j,1]
            +RePsi[i-1,j,1]))
            ai2 = dm2*(RePsi[i,j,1]*(ImPsi[i,j+1,1]+ImPsi[i,j-1,1]) - ImPsi[i,j,1]*(RePsi[i,j+1,1]
            +ImPsi[i,j-1,1]))
            Er = w[k]*w[p]*con*(a1 + a2 + a3)
            Ei = w[k]*w[p]*con*(ai1 + ai2)
            k = k + 1
        end
        p = p + 1
    end

    # Normalize
    Er = Er / Ptot_i
    Ei = Ei / Ptot_i

    if abs(Er) <= 1e-20
        Er = 0.0
    end

    println(out11, "$n $Er")

    tmp = abs(Er / Eri - 1)
    if tmp != 0.0
        Erel = log10(tmp)
    end

    println(out12, "$n $Erel")
    print(Er)
    print(Erel)
end

function Output(n::Int, out2)

    global Rho, rho_1, rho_2, corr, v

    s = "run.00001.csv"

    # Create a unique file name for each time step
    if n < 10
        s = s[1:8] * string(n)
    elseif n < 100 && n > 9
        s = s[1:7] * string(n)
    elseif n < 1000 && n > 99
        s = s[1:6] * string(n)
    elseif n < 10000 && n > 999
        s = s[1:5] * string(n)
    elseif n < 100000 && n > 9999
        s = s[1:4] * string(n)
    end

    # Print probability vs position
    if choice == 0
        if n == 1
            open(s, "w") do out1
                for i in 1:N2
                    for j in 1:N2
                        if Rho[i, j] < 1e-20
                            Rho[i, j] = 1e-20
                        end
                        write(out1, "$(Rho[i, j])\n")
                    end
                    write(out1, "\n")
                end
            end
        end
        open(s, "w") do out1
            for i in 1:N2
                for j in 1:N2
                    if Rho[i, j] < 1e-20
                        Rho[i, j] = 1e-20
                    end
```

```
                write(out1, "$(Rho[i, j])\n")
            end
            write(out1, "\n")
        end
    end
elseif choice == 1
    open(s, "w") do out1
        for i in 1:N1
            if rho_1[i] < 1e-20
                rho_1[i] = 1e-20
            end
            if rho_2[i] < 1e-20
                rho_2[i] = 1e-20
            end
            write(out1, "$i $(rho_1[i]) $(rho_2[i])\n")
        end
    end
elseif choice == 2
    open(s, "w") do out1
        for i in 1:N2
            if rho_1[i] < 1e-20
                rho_1[i] = 1e-20
            end
            if rho_2[i] < 1e-20
                rho_2[i] = 1e-20
            end
            write(out1, "$i $(corr[i])\n")
        end
    end
elseif choice == 3
    open(s, "w") do out1
        for i in 1:N1
            sol = rho_2[i] + rho_1[i]
            if sol < 1e-20
                sol = 1e-20
            end
            write(out1, "$i $sol\n")
        end
    end
end

if n == 1
    # Print out potential
    for i in 1:10:N1, j in 1:10:N2
        println(out2, v[i,j])
    end

    # Print out two-particle probability & correlation at t = 1
    out7a = open("Rho_0.dat", "w")
    out3a = open("C_0T.dat", "w")
    for i in 1:10:N1, j in 1:10:N2
        sol = Rho[i,j]
        if sol < 1.2e-20
            sol = 0.
        end
        println(out7a, sol)
        if rho_1[i] != 0. && rho_2[j] != 0.
            sol = log10(abs(Rho[i,j]/rho_1[i]/rho_2[j]))
        else
            sol = 0.
        end
        println(out3a, sol)
    end
    close(out7a)
    close(out3a)
end

if n == div(Nt,2)
    out7b = open("Rho_.5.dat", "w")
    out3b = open("C_.5T.dat", "w")
    for i = 1:10:N1, j = 1:10:N2
        sol = Rho[i,j]
        if sol < 1.2e-20
            sol = 0.0
        end
        println(out7b, "%e", sol)
        if (rho_1[i] != 0.0) && (rho_2[j] != 0.0)
```

```
                sol = log10(abs(Rho[i,j]/rho_1[i]/rho_2[j]))
            else
                sol = 0.0
            end
            println(out3b, "%e", sol)
        end
        println(out7b, "\n")
        println(out3b, "\n")
        close(out7b)
        close(out3b)
    end

    if n == Nt
        out7c = open("Rho_1.dat", "w")
        out3c = open("C_T.dat", "w")
        for i = 1:10:N1, j = 1:10:N2
            sol = Rho[i,j]
            if sol < 1.2e-20
                sol = 0.0
            end
            println(out7c, "%e", sol)
            if (rho_1[i] != 0.0) && (rho_2[j] != 0.0)
                sol = log10(abs(Rho[i,j]/rho_1[i]/rho_2[j]))
            else
                sol = 0.0
            end
            println(out3c, "%e", sol)
        end
        println(out7c, "\n")
        println(out3c, "\n")
        close(out7c)
        close(out3c)
    end

end


function Potential()
    global dx, alpha, vmax, N1, N2, v

    tmp = dx^2 / alpha^2 / 2.0
    # Calculate gaussian potential
    for i in 0:(N1 + 1)
        for j in 0:(N2 + 1)
            v[i + 1, j + 1] = -vmax * exp(-tmp * (i - j)^2)
        end
    end

    # Calculate square potential
    # tmp = alpha/dx;
    # if abs(i-j)*dx <= alpha
    #     v[i,j] = vmax
    # else
    #     v[i,j] = 0.0
    # end
end

 function Probability(n)
    global N1, N2, Rho, w, Ptot_i, Ptot, rho_1, rho_2, corr, SumRho, out8, out10, out13

    for i in 0:(N1 + 1)
        rho_1[i + 1] = 0.0
        rho_2[i + 1] = 0.0
        corr[i + 1] = 0.0
    end

    Ptot = 0.0
    Prel = 0.0
    p = 1
    for i in 1:N1
        k = 1
        if p % 3 == 0
            p = 1
        end
        for j in 1:N2
            if k % 3 == 0
                k = 1
```

```
            end
            Ptot += w[k] * w[p] * Rho[i, j]
            k += 1
        end
        p += 1
    end

    if n == 1
        Ptot_i = Ptot
    end

    for i in 1:N1
        for j in 1:N2
            Rho[i, j] = Rho[i, j] / Ptot
        end
    end

    p = 1
    for i in 1:N1
        k = 1
        if p % 3 == 0
            p = 1
        end
        for j in 1:N2
            if k % 3 == 0
                k = 1
            end
            rho_1[i] += w[k] * Rho[i, j]
            rho_2[i] += w[k] * Rho[j, i]
            k += 1
        end

        SumRho[i] = rho_1[i] + rho_2[i]
        if abs(SumRho[i]) < 1.0e-20
            SumRho[i] = 0.0
        end
        if i % 10 == 0
            # Replace 'out8'
            # fprintf(out8, "%e\n", SumRho[i])
        end
        p += 1
    end
    # fprintf(out8, "\n")

    tmp = abs((Ptot / Ptot_i) - 1.0)
    if tmp != 0.0
        Prel = log10(tmp)
    end
    # Replace 'out10'
    # fprintf(out10, "%d %e\n", n, Prel)

    if n == div(Nt, 2)
        for i in 1:5:N1
            j = N1 + 1 - i
            x = abs(i - j)
            if rho_1[i] != 0.0 && rho_2[j] != 0.0
                corr[i] = Rho[i, j] / rho_1[i] / rho_2[j]
            end
            if rho_1[j] != 0.0 && rho_2[i] != 0.0
                corr[i] += Rho[j, i] / rho_1[j] / rho_2[i]
            end
            if corr[i] != 0.0
                corr[i] = log10(abs(corr[i]))
            end
            # Replace 'out13'
            # fprintf(out13, "%lf %e\n", x, corr[i])
        end
        # fprintf(out13, "\n")
    end
end

function SolveSE(out2, out11, out12)
    global dt, v, ImPsi, RePsi, N1, N2, Nt, Rho, con2, dtx, dm12, dm22, nprint

    for n in 1:Nt
        for i in 2:(N1-1)
            for j in 2:(N2-1)
```

```
                a2 = dt * v[i, j] * ImPsi[i, j, 1] + con2 * ImPsi[i, j, 1]
                a1 = dm12 * (ImPsi[i + 1, j, 1] + ImPsi[i - 1, j, 1]) +
                     dm22 * (ImPsi[i, j + 1, 1] + ImPsi[i, j - 1, 1])
                RePsi[i, j, 2] = RePsi[i, j, 1] - dtx * a1 + 2.0 * a2

                if n % nprint == 0 || n == 1 || n == div(Nt, 2) || n == Nt
                    Rho[i, j] = RePsi[i, j, 1] * RePsi[i, j, 2] +
                                ImPsi[i, j, 1] * ImPsi[i, j, 1]
                end
            end
        end

        if n % nprint == 0 || n == 1 || n == div(Nt, 2) || n == Nt
            Probability(n)
        end

        for i in 2:(N1-1)
            for j in 2:(N2-1)
                a2 = dt * v[i, j] * RePsi[i, j, 2] + con2 * RePsi[i, j, 2]
                a1 = dm12 * (RePsi[i + 1, j, 2] + RePsi[i - 1, j, 2]) +
                     dm22 * (RePsi[i, j + 1, 2] + RePsi[i, j - 1, 2])
                ImPsi[i, j, 2] = ImPsi[i, j, 1] + dtx * a1 - 2.0 * a2
            end
        end

        if n % nprint == 0 || n == 1
            Energy(n, out11, out12)
        end

        if n % nprint == 0 || n == 1 || n == div(Nt, 2) || n == Nt
            Output(n, out2)
        end

        for i in 1:N1
            for j in 1:N2
                ImPsi[i, j, 1] = ImPsi[i, j, 2]
                RePsi[i, j, 1] = RePsi[i, j, 2]
            end
        end
    end
end

function main()
    code_block = begin
    # open files
    out2  = open("V.dat", "w")
    out3a = open("C_0T.dat", "w")
    out3b = open("C_.5T.dat", "w")
    out3c = open("C_T.dat", "w")
    out7a = open("Rho_0.dat", "w")
    out7b = open("Rho_.5.dat", "w")
    out7c = open("Rho_1.dat", "w")
    out8  = open("SumRho.dat", "w")
    out10 = open("logP_t.dat", "w")
    out11 = open("E_t.dat", "w")
    out12 = open("logE_t.dat", "w")
    out14 = open("params.dat", "w")
    out13 = open("c_x.dat", "w")

    # print inital values sto the parameters file
    println(out14, "k1 = ", k1)
    println(out14, "k2 = ", k2)
    println(out14, "dx = ", dx)
    println(out14, "dt = ", dt)
    println(out14, "sig = ", sig)
    println(out14, "alpha = ", alpha)
    println(out14, "x01 = ", x01)
    println(out14, "x02 = ", x02)
    println(out14, "N1 = ", N1)
    println(out14, "N2 = ", N2)
    println(out14, "m1 = ", m1)
    println(out14, "m2 = ", m2)
    println(out14, "vmax = ", vmax)
    println(out14, "nprint = ", nprint)
    println(out14, "xDim = ", xDim)
    println(out14, "Nt = ", Nt)
```

```
    # Call the functions
    Potential()
    Initialize(out14)
    SolveSE(out2, out11, out12)
    end
    measure_time = @elapsed code_block
    println(out14, "\nElapsed time is:  \n", measure_time)


    close(out8)
    close(out10)
    close(out11)
    close(out12)
    close(out14)
end

main()
```

Listing A.2: Code for animating two particle schrodinger equation in 1D

```
using CSV, Plots, DataFrames

function animate_runs()
    anim = @animate for i in 300:300:40000
        df = CSV.read("run.$(lpad(i, 5, '0')).csv", DataFrame, header=false)
        p = plot(df[:, 1], df[:, 2], label="rho1", ylim = (0,12), linestyle=:dash, xlabel="Position",
        ylabel="Probability",legend = :topright)
        plot!(df[:, 1], df[:, 3], label="rho2", linestyle=:dash)
        annotate!(50, 10,string("Run ", i), 10)
        p
    end
    gif(anim, "rho_1 (2D).gif", fps=15)
end

animate_runs()
```

Listing A.3: Code for animating two particle schrodinger equation in 1D

```
using LaTeXStrings
using QuadGK  # package for numerical integration


# define constants
Z = 2                 # Nuclear Charge
a0 = 0.5291772108     # Bohr radius
# a0 = 1
R_max = 10

# define the wavefunction (found from Hydrogen, so wavefunction should simply be the product
# of two hydrogenic wavefunctions for helium - not considering the repulsion of the electron
# electron interaction)

function psi(r1, r2)
    R1 = 1/(sqrt(pi))*(Z/a0)^3/2 * exp(-(Z*r1)/a0)
    R2 = 1/(sqrt(pi))*(Z/a0)^3/2 * exp(-(Z*r2)/a0)
    R = R1 * R2
    # return (2/(Z))^3/2 * exp(-r1/Z) * exp(-r2/Z)
    return R
end

# define the probability density
function P(r1, r2)
    return abs2(psi(r1, r2))
end

# define the integrand for P(r2)
function integrand(r1, r2)
    return P(r1, r2) * r1^2
end

# define the function to calculate P(r2) using numerical integration
```

```
function calculate_P_r2(r2)
    integral, err = quadgk(r1 -> integrand(r1, r2), 0, R_max)
    return 4 * pi * r2^2 * integral
end

# calculate P(r2) for a range of r2 values
r2_values = collect(range(0, stop=2, length=100))
P_r2_values = [calculate_P_r2(r2) for r2 in r2_values]

# plot P(r2) vs r
using Plots
plot(r2_values, P_r2_values, xlabel= L"r", ylabel= L"P(r_2)",
title= "Radial probability for two electron system",
titlefontsize= 10,legend=false)
savefig("Radial Probability plot.png")
```

Listing A.4: Code for calculating variational energies with adjustable parameter Z

```
using Plots

# Graph of variational energies for helium atom as a function
# of the adjustable parameter Z , which represents the
# effective nuclear charge felt by the electrons

function variational_energy(Z)
    E = 13.6*(2*Z^2 - (27/4) * Z)
    return E
end

z_values = 0:0.1:5
e_values = [variational_energy(z) for z in z_values]

plot(z_values, e_values, xlabel = "Z", ylabel = "Energy", legend = false)
# savefig("variational energies He using Z , which represents the
# effective nuclear charge felt by the electrons.")
```

Listing A.5: Code for visualising 2D electron scattering

```
using Plots, LinearAlgebra, LaTeXStrings

#Storage variables for schrodinger wave function array
Base.@kwdef mutable struct schrodinger_eq

           = Array{Float64, 2}
        A = Array{Float64, 2}
        B = Array{Float64, 2}
        d = Array{Float64, 2}
        s = Array{Float64, 2}
end

# General wave function and domain parameters
Base.@kwdef mutable struct params
        x1::Array{Float64}
        x2::Array{Float64}
        dx::Float64
        t::Float64
        dt::Float64
           ::Float64
        N::Int64 = length(x)
        kx1::Float64
        kx2::Float64
end

# Struct for saving and buildingthe barrier/double slit dimensions
Base.@kwdef mutable struct draw_slits
    pos::Float64
    width::Float64
    slit_size::Float64
end

# Creates coefficient matrices by taking in the
function diff_matrix(diag,offdiag, Ns)
```

```
    A = zeros(ComplexF64, Ns, Ns)
    A[diagind(A,0)] .= diag
    A[diagind(A,1)] .= A[diagind(A,-1)] .= offdiag
    return A
end

# Generates the two dimensional initial wave function distribution
function init_ (x1, x2, x1_0, x2_0, k1x_0, k2x_0,    , N)

       = zeros(ComplexF64, N, N)

    for i in 1:N
        for j in 1:N
            Nx1 = exp(-((x1[i]-x1_0)^2) / (2*   ^2)) * exp(im*k1x_0*x1[i])
            Nx2 = exp(-((x2[j]-x2_0)^2) / (2*   ^2)) * exp(im*k2x_0*x2[j])
               [i,j] = Nx1 * Nx2
            #    [i,j] = exp(-((x1[i]-x1_0)^2) / (4*   ))*exp(-((x2[j]-x2_0)^2) / (4*   ))*exp(im*k1x_0*x1[i] + im*k2x
        end
    end
    return transpose(   )
end



function U_accelerate(pos, par::params)
    V = zeros(par.N,par.N)
    pos = pos/maximum(par.x) * par.N
    pos = Int32(round(pos))

    V[ : , pos:par.N] .= -90

    return V
end

# Creates double slit potential shape
function make_U_doubleslit(par::params, draw::draw_slits)

    V = zeros(par.N,par.N)
    pos_main = trunc(Int, (draw.pos*(par.N)) / maximum(par.x1))
    pos = trunc(Int, (par.N)*.5)+1
    gap = round(Int, ((par.N-1)*draw.slit_size/maximum(par.x1))/2)
    width = Int32(trunc(draw.width*(par.N-1)/maximum(par.x1)))

    V[ : , pos_main:pos_main+width] .= 1000
    V[pos+gap:pos+3gap, pos_main:pos_main+width] .= 0
    V[pos-3gap:pos-gap, pos_main:pos_main+width] .= 0
    return V
end


function make_atomic_potential( par::params)
    Z = 2
    l1 = 0
    l2 = 0

    # V = zeros(ComplexF64, par.N, par.N)
    V = zeros(par.N)

    for i in 1:par.N
        V[i] = - Z/par.x1[i] + l1*(l1+1)/(2*par.x1[i]^2) - Z/par.x2[i] + l2*(l2+1)/(2*par.x2[i]^2)
    end
    return V
end

# Finds local maxima of 1d array
function get_peaks(fringe, x)
    peaks = Float64[]
    for i in 2:1:length(fringe)
        if fringe[i] > 0.01 && fringe[i] < fringe[i-1] && fringe[i-1] > fringe[i-2]
            push!(peaks, x[i-1])
        end
    end
    for i in 1:1:length(peaks)
        if peaks[i] == 7.5
            peaks[i] = 0
        elseif peaks[i] > 7.5
            peaks[i] = round(peaks[i]-7.5, digits=6)
```

```
        else
            peaks[i] = round(-(7.5-peaks[i]), digits=6)
        end
    end
    return peaks
end

# Draws double slits for plotting
function draw_potential_slits_barrier(draw::draw_slits, par::params)
    p = draw.pos
    w = draw.width
    draw_slit(pos, width, ymax, ymin) = Shape([(pos, ymin),(pos+width, ymin),(pos+width, ymax),(pos, ymax)])
    draw_top = draw_slit(p, w, maximum(par.x2), (maximum(par.x2)/2)+1.5draw.slit_size)
    draw_mid = draw_slit(p, w, (maximum(par.x2)/2)-.5draw.slit_size, (maximum(par.x2)/2)+.5draw.slit_size)
    draw_bottom = draw_slit(p, w, (maximum(par.x2)/2)-1.5draw.slit_size, minimum(par.x2))

    draw_barrier = draw_slit(p, w, maximum(par.x2), minimum(par.x2))
    return draw_top, draw_mid, draw_bottom, draw_barrier
end

# This the main part of the program and propagates the given wave function through time
# and updates the solution at each step.

# It requires that the domain and wavefunction parameters are initialized...
# it requires an initial wavefunction and the coefficient matrices for the kinetic operator..
# as well as the 2d potential operators.
# It also requires that the dimensions of the barrier are specified. If none, just set draw_slits parameters to 0.

function solver(par::params, sch::schrodinger_eq, draw::draw_slits)
    local pos_real
    c = (sch.B / sch.A )
    norm_init = sum(abs2.(sch.  ))
    Nt = length(0:par.dt:4.0)
    copy_fringe = 0
    @gif for i in 1:Nt

        par.t = (i-1)*par.dt

        # Split step ADI method used for solving TDSE
        for x in 2:par.N-1
            a = sch.  [ : , x ]
            sch.  [ : , x ] =  c *  a
            sch.  [end, x] = sch.  [1, x] = 0
        end

        for x in 2:par.N-1
            a = sch.  [x, : ]
            a[1] = a[end] = 0
            sch.  [x, : ] = c * a
            sch.  [x, : ] = (sch.  [x, : ]) .* (sch.s[ x , : ] ./ sch.d[x, : ])
        end

        draw_top, draw_mid, draw_bottom, draw_barrier = draw_potential_slits_barrier(draw, par)

        pos_real = 29
        pos = Int64(floor((pos_real*par.N) / maximum(par.x1)))

        fringe = abs.(sch.  [ : , pos ])

        # Generates 2d heatmap

        #plot_2d = plot(draw_barrier, alpha=0.5, c=:grey, label="")
        plot_2d = heatmap(par.x1, par.x2, (abs.(sch.  )),  colorbar=false, legend=false, c=:thermal)
        plot_2d = plot!([pos_real], seriestype=:vline, linestyle =:dash, label="")
        plot_2d = plot!(draw_top, alpha=0.5, c=:grey, label="")
        plot_2d = plot!(draw_bottom, alpha=0.5, c=:grey, label="")
        plot_2d = plot!(draw_mid, alpha=0.5, c=:grey, label="")
        plot_2d = annotate!(20, 50,Plots.text(string("Time:" ,round(par.t, digits=5)), 10, :white) )
        plot!(border=false, axis=false)
        plot!(size=(400,400))


        # Attempts to update the plot in realtime. Can be removed without any loss of function.
        if mod(i, 15) == 0
            plot_fringe = plot(par.x1 ,fringe, ylim=(0, .3), ylabel=L"Transmitted \;\; Intensity",
                               label="", title = L"Intensity \;\;at\;\; l=3.75")
            plot_3d = surface(abs.(sch.  ), c =cgrad(:berlin, scale = :exp, rev=true), zlim=(0,0.8))
```

```
            plot(plot_2d, plot_3d,  size = (400, 800), layout = (2,1))

            display(current())
        end

        # Detects when the amplitude peak begins to fall. When it does, it breaks from the solver function and retu
        # the error between the expected and observed wavenumber

        if ((maximum(fringe) - maximum(copy_fringe)) < -.0000001) && maximum(fringe) > 0.1

            display(current())

            ######## MAKE FUNCTION
            detector_distance = (pos_real-(draw.pos+draw.width))
            peaks  = get_peaks(fringe, par.x1)
            #println(peaks)
            index = Int32(floor(length(peaks)/2)+1)
            m1 = abs(peaks[index+1]-peaks[index])
            # Calculates the observed wavenumber using the location of the peaks, the slit_size*2
            # and distance between the exit of the slits and the detector
            observed_k = 2  /((2*draw.slit_size)*sin(tan(m1/detector_distance)))
            println("Observed k = ", observed_k, "\t Expected k = ", par.kx1)
            println(peaks[index+1]-peaks[index])
            ################################
            plot()
            est_k = round(observed_k, digits=4)
            plot_fringe = plot(par.x1 ,fringe, ylim=(0, .3), ylabel=L"Transmitted \;\; Intensity", label="",
                               title = L"Intensity \;\;at\;\; l=3.75", xticks = (0:1.25:15, -7.5:1.25:7.5))
            plot_fringe= annotate!(2.9,.24,latexstring("\$\\ Est.k_{x_0}={$(est_k)}\$"), 12)
            plot_fringe= annotate!(2.5,.22,latexstring("\$\\ Init.k_{x_0}={$(par.kx1)}\$"), 12)

            plot(plot_2d, plot_fringe, size=(800, 400))
            display(current())
            plot()
            sleep(10)
            return abs.(par.kx1-observed_k)/par.kx1
            @goto escape_label
        end

        copy_fringe = copy(fringe)

    end every 50

    @label escape_label

end

for i in 1:1
        global d, s
        dx = .1

        t = 0
        dt= 0.5*dx^2
        x1 = x2 = collect(0:dx:30)
        N = length(x1)
        println("dx = ", dx, "\t    dt= ", dt)

           = 1

        k1x_0 = 7
        k2x_0 = 7
        x1_0 = 1
        x2_0 = 1
           = (4dx^2) / dt
        # Initialize parameters and construct kinetic coefficient matrices
        par = params(x1, x2, dx, t, dt,    , N, k1x_0, k2x_0)
        B = diff_matrix(1-2im/  , im/  , par.N)
        A = diff_matrix(1+2im/  , -im/  , par.N)


        slit_size = 0  #along y-direction
        slit_width = 0 #along x-direction
        x_pos = 30-slit_width
        draw = draw_slits(x_pos, slit_width, slit_size)

        # V = make_U_doubleslit(par, draw)
        # V = V + U_accelerate(5, par)
```

```
        V = make_atomic_potential(par)

        d = exp.(im*dt.*V/2)
        s = exp.(-im*dt.*V/2)

           = init_ (x1, x2, x1_0, x2_0, k1x_0, k2x_0,    , par.N)

        sch = schrodinger_eq(   , A, B, d, s)

        error = solver(par, sch, draw)
        println("Error %: ", error)
end
```

Bibliography

[1] Attila Askar and Ahmet S Cakmak. "Explicit integration method for the time-dependent Schrodinger equation for collision problems". In: *The Journal of Chemical Physics* 68.6 (1978), pp. 2794–2798.

[2] CL Davis and EN Maslen. "On exact analytical solutions for the few-particle Schrödinger equation. II. The ground state of helium". In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 384.1786 (1982), pp. 89–105.

[3] Vladimir Fock. "Näherungsmethode zur Lösung des quantenmechanischen Mehrkörperproblems". In: *Zeitschrift für Physik* 61 (1930), pp. 126–148.

[4] Abraham Goldberg, Harry M Schey, and Judah L Schwartz. "Computer-generated motion pictures of one-dimensional quantum-mechanical transmission and reflection phenomena". In: *American Journal of Physics* 35.3 (1967), pp. 177–186.

[5] Douglas R Hartree. "The wave mechanics of an atom with a non-Coulomb central field. Part I. Theory and methods". In: *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 24. 1. Cambridge university press. 1928, pp. 89–110.

[6] Daniel Hockey. "Numerical Methods for Solving the Time-Dependent Schrodinger Equation". Final Year Project Report. 2020.

[7] Egil A Hylleraas. "Neue berechnung der energie des heliums im grundzustande, sowie des tiefsten terms von ortho-helium". In: *Zeitschrift für Physik* 54.5-6 (1929), pp. 347–366.

[8] Alexander Kramida, Yuri Ralchenko, Joseph Reader, et al. "NIST atomic spectra database". In: *NIST standard reference database* 78 (2018).

[9] John L Richardson. "Visualizing quantum scattering on the CM-2 supercomputer". In: *Computer Physics Communications* 63.1-3 (1991), pp. 84–94.

[10] Faisal Saied. *Numerical techniques for the solution of the time-dependent Schroedinger equation and their parallel implementation*. Yale University, 1990.

[11] PB Visscher. "A fast explicit algorithm for the time-dependent Schrödinger equation". In: *Computers in Physics* 5.6 (1991), pp. 596–598.