Numerical Analysis of Schrödinger Equation

Term Project Report

Ahmet Burak Yıldırım* **& Yiğit Yaman†**

Mechanical Engineering Department, İ.D. Bilkent University, Ankara 06800 Turkey

Abstract: In this paper, the results of a numerical analysis of the time-dependent and time-independent Schrödinger equation using finite difference methods are presented. A range of potentials is considered, and the behavior of quantum systems in one and two dimensions is studied. The numerical algorithms for solving the Schrödinger equation are developed in MATLAB® and applied to a variety of problems in quantum mechanics. Through this work, a deeper understanding of the behavior of quantum systems is gained, and the effectiveness of the numerical methods is demonstrated.

Keywords: Schrödinger equation • Quantum mechanics • Partial differential equations

1. Introduction

The Schrödinger equation is a fundamental equation in quantum mechanics that describes the time evolution of a quantum mechanical system. It plays a central role in our understanding of the behavior of matter at the atomic and subatomic scales and has wide-ranging applications in fields such as chemistry, materials science, and nanotechnology. Despite its importance, the Schrödinger equation is often difficult to solve analytically, especially for complex systems or when the potential energy is time-dependent. As a result, there is a solid motivation to develop numerical methods for solving the Schrödinger equation accurately and efficiently [1–4].

This project addresses this need by developing and implementing numerical algorithms for solving the time-dependent and time-independent Schrödinger equation using finite difference methods. Our goal is to study the behavior of quantum systems in one and two dimensions and to understand how the choice of potential energy affects the solutions of the Schrödinger equation. We will apply our algorithms to various problems in quantum mechanics, including the study of well-known quantum systems, such as the harmonic oscillator and the quantum well, and more complex systems, such as those involving non-trivial potentials.

The significance of our analysis lies in its potential to provide new insights into the behavior of quantum systems and support the development of new technologies that rely on manipulating quantum states. By developing

* *E-mail: ahmet.yildirim@bilkent.edu.tr*
† *E-mail: yigit.yaman@bilkent.edu.tr*

accurate and efficient numerical algorithms, we hope to make it possible to study more complex systems and explore a broader range of examples in quantum mechanics.

2. Numerical Modeling

In our numerical analysis of the Schrödinger equation, we consider both the time-dependent and time-independent versions of the equation. The most general form of the time-dependent Schrödinger equation is given by:

$$i\hbar\frac{\partial}{\partial t}\psi(\mathbf{r},t) = \left(-\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r},t)\right)\psi(\mathbf{r},t) \tag{1}$$

where $\mathbf{r}$ is the position vector, $\psi(\mathbf{r},t)$ is the wave function, $\hbar$ is the reduced Planck constant, m is the mass of the particle, ∇^2 is the Laplacian operator, and $V(\mathbf{r},t)$ is the potential energy. In the context of this project, the potential energy is assumed to vary only spatially $V(\mathbf{r},t) = V(\mathbf{r})$, yielding the following time-dependent Schrödinger equation:

$$i\hbar\frac{\partial}{\partial t}\psi(\mathbf{r},t) = \left(-\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r},t)\right)\psi(\mathbf{r},t) \tag{2}$$

and the complementary time-independent form:

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

where $\psi(\mathbf{r})$ is the stationary wave function, E is the energy of the system.

For both the time-independent and time-dependent Schrödinger equation, three-point central finite difference methods are used to approximate the spatial derivatives given by the Laplacian operator, given in 1D and 2D, respectively:

$$\nabla^2\psi(x) \approx \frac{\psi(x-\Delta x) - 2\psi(x) + \psi(x+\Delta x)}{(\Delta x)^2} \tag{4}$$

$$\nabla^2\psi(x,y) \approx \frac{\psi(x-\Delta x,y) - 2\psi(x,y) + \psi(x+\Delta x,y)}{(\Delta x)^2} + \frac{\psi(x,y-\Delta y) - 2\psi(x,y) + \psi(x,y+\Delta y)}{(\Delta y)^2} \tag{5}$$

The domain of the problem is discretized, and the wave function and potential energy at each point in the discretized domain are represented as a vector (in 1D) or a matrix (in 2D). Matrix operations are used to update the wave function at each time-step or to solve for the energy eigenvalues and eigenvectors in the time-independent case.

In the context of the boundary conditions, regardless of the interior potential $V(\mathbf{r})$, reflective boundary conditions are applied to ensure the continuity of the wave function at the boundaries of the computational domain.

This is done by reflecting the wave function at the boundaries, resulting in a zero value of the wave function at the boundaries $\big($for 1D computational domain x $\in$ [a, b] : $\psi(x = a, t) = \psi(x = b, t) = 0\big)$. This is a specific case of the more general class of Dirichlet boundary conditions, where the value of the function of interest is specified at the boundaries. For instance in 1D, for an arbitrary interior potential $V^*(x)$, the total potential term $V^*(x)$ is given by

$$V(x) = \begin{cases} V^*(x), & x \in (a, b) \\ \infty, & x \notin (a, b) \end{cases} \tag{6}$$

With this, the probability current density $j(\psi)$, a mathematical quantity describing the flow of probability, which is:

$$j(\psi) = \frac{i\hbar}{2m}(\psi\nabla\psi^* - \psi^*\nabla\psi) \tag{7}$$

where ψ^* is the conjugate of ψ, can also be shown to vanish at the reflective boundaries [4]:

$$j(\psi)|_{x=a,b} = \frac{i\hbar}{2m}(\psi\nabla\psi^* - \psi^*\nabla\psi)_{x=a,b} = 0 \tag{8}$$

Eqn. (8) shows that the probability of finding the particle and the flux associated with it at the infinite boundaries are both zero. Therefore, in the case of quantum tunneling, which is shown later, the particle must not be trapped within an infinite well to exhibit non-classical dynamics.

The initial conditions of the wave function are specified either as a specific energy eigenstate in the time-independent case or a function of space and time to initialize the wave function in the simulations.

For TISE, Eqn. (3), the initial guess of the wave function has the following form:

$$\psi(\mathbf{r}, t) = A \tag{9}$$

where A is the normalization constant to ensure the total probability $|\psi(\mathbf{r}, t)|$ satisfies:

$$\int_{Domain} |\psi(\mathbf{r}, t)|^2 d\mathbf{r} = 1 \tag{10}$$

On the other hand, TDSE, Eqn. (2), is initialized with the Gaussian wave packet [5]:

$$\psi(\mathbf{r}, t) = Ae^{-\frac{(\mathbf{r}-\mathbf{r_0})^2}{2\sigma^2}} e^{-i(\mathbf{k}\cdot\mathbf{r})} \tag{11}$$

where $\mathbf{r_0}$ denotes the initial position of the center of the wave packet, σ determines the width of the wave packet, and $\mathbf{k}$ is the wave vector associated with the momentum of the wave packet. For both TISE and TDSE, the normalization constant A is updated to satisfy Eqn. (10) at every iteration or time-step, respectively.

As the context of this project is limited to the numerical analysis of TDSE, Eqn. (2) and TISE, Eqn. (3), in one and two dimensions, the spatial discretization of the computational domain must include one and two indices,

respectively. On the other hand, iterative or temporal discretization is shown with a single index, k. The current iteration or time-step of the wave function is denoted with ψ^k, and the next iteration or time-step of the wave function is denoted with ψ^{k+1}. Regarding spatial discretization, the following stencils denote spatial indices in one and two dimensions, respectively.

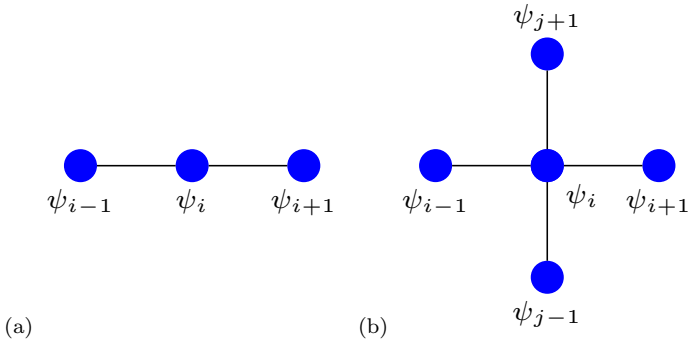

Figure 1: Stencils for spatial discretization of the wave function ψ in 1D: (a), and in 2D: (b).

2.1. Time-independent Schrödinger Equation (TISE)

The time-independent Schrödinger equation (TISE), Eqn. (3), describes a non-relativistic particle's behavior in a given potential energy field and gives information about the allowed energy states of the system [2, 3]. It can be mathematically represented as an *elliptic* partial differential equation (PDE), characterized by unique steady-state solutions for a given set of initial and boundary conditions [6].

TISE plays a vital role in the understanding of various physical systems, particularly in atomic, molecular, and solid-state physics. For example, TISE describes the atom's electronic structure and energy levels in atoms, while in molecules, it gives information about the bond lengths, angles, and vibrational modes. In solid-state physics, TISE is used to study the electronic properties of materials such as semiconductors, superconductors, and insulators [3].

Due to the complexity of potentials in many physical systems, analytical solutions to TISE are not always possible. Therefore, numerical methods are used to approximate the solutions to TISE. The numerical analysis of elliptic PDEs yields some methods with fast convergence rates for sufficiently good approximations, making it possible to study the influence of various parameters on the system [6].

2.1.1. One dimensional solution

In 1D, the position vector becomes $\mathbf{r} = (x)$ and TISE, Eqn. (3), reduces to the following:

$$-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}\psi(x) + V(x)\psi(x) = E\psi(x) \tag{12}$$

where a new potential can be written as $V'(x) = V(x) - E$ since E is a constant-valued energy state. By renaming the potential $V'(x)$ to $V(x)$, a more reduced form of Eqn. (12) can be obtained:

$$-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}\psi(x) + V(x)\psi(x) = 0 \tag{13}$$

In this project, to numerically evaluate Eqn. (13), Jacobi and Gauss-Seidel methods are considered. Spatial discretization of Eqn. (13) using the finite differencing scheme given for 1D, see Eqn. (4), yields the following:

$$-\frac{\hbar^2}{2m}\left[\frac{\psi_{i-1} - 2\psi_i + \psi_{i+1}}{(\Delta x)^2}\right] + V_i\psi_i = 0 \tag{14}$$

Rearranging Eqn. (14) to solve for ψ_i yields:

$$\psi_i = \frac{\hbar^2\left(\psi_{i-1} + \psi_{i-1}\right)}{2\hbar^2 + 2V_i m(\Delta x)^2} \tag{15}$$

According to the Jacobi method, the approximate wave function at the next iteration, ψ^{k+1}, is constructed using the wave function only at the current iteration, ψ^k. Hence, the resulting equation is solved at every iteration in the Jacobi method:

$$\boxed{\psi_i^{k+1} = \frac{\hbar^2\left(\psi_{i-1}^{k} + \psi_{i+1}^{k}\right)}{2\hbar^2 + 2V_i m(\Delta x)^2}} \tag{16}$$

where ψ_i^{k+1} is the wave function evaluated at the next iteration.

On the other hand, the Gauss-Seidel method suggests constructing ψ^{k+1} based on the latest approximations of the wave function, i.e., using the approximate wave function at the next time-step ψ_i^{k+1} whenever applicable. Considering the node (i+1) is updated after the node (i), the resulting equation is solved at every iteration in the Gauss-Seidel method:

$$\boxed{\psi_i^{k+1} = \frac{\hbar^2\left(\psi_{i-1}^{k+1} + \psi_{i+1}^{k}\right)}{2\hbar^2 + 2V_i m(\Delta x)^2}} \tag{17}$$

where ψ_i^{k+1} and ψ_{i-1}^{k+1} are the wave functions evaluated at the next iteration.

2.1.2. Two dimensional solution

In 2D, the position vector becomes $\mathbf{r} = (x, y)$ and TISE, Eqn. (3), reduces to the following:

$$-\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)\psi(x,y) + V(x,y)\psi(x,y) = E\psi(x,y) \tag{18}$$

where the a new potential similarly can be written as $V'(x,y) = V(x,y) - E$ since E is a constant-valued energy state. By renaming the potential $V'(x,y)$ to $V(x,y)$, a more reduced form of Eqn. (18) can be obtained:

$$-\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)\psi(x,y) + V(x,y)\psi(x,y) = 0 \tag{19}$$

Again, Jacobi and Gauss-Seidel methods are used to evaluate Eqn. (19). Spatial discretization of Eqn. (19) using the finite differencing scheme given for 2D, see Eqn. (5), yields the following:

$$-\frac{\hbar^2}{2m}\left[\frac{\psi_{i-1,j}-2\psi_{i,j}+\psi_{i+1,j}}{(\Delta x)^2}+\frac{\psi_{i,j-1}-2\psi_{i,j}+\psi_{i,j+1}}{(\Delta y)^2}\right]+V_{i,j}\psi_{i,j}=0 \tag{20}$$

Rearranging Eqn. (20) to solve for ψ_i yields:

$$\psi_{i,j}=\frac{\hbar^2\left[(\Delta y)^2\left(\psi_{i-1,j}+\psi_{i+1,j}\right)+(\Delta x)^2\left(\psi_{i,j-1}+\psi_{i,j+1}\right)\right]}{2\hbar^2\left[(\Delta x)^2+(\Delta y)^2\right]+2V_{i,j}m(\Delta x)^2(\Delta y)^2} \tag{21}$$

After constructing the approximate wave function at the next iteration, ψ^{k+1}, using the wave function only at the current iteration, the resulting equation is obtained, which is solved at every iteration in the Jacobi method:

$$\boxed{\psi_{i,j}^{k+1}=\frac{\hbar^2\left[(\Delta y)^2\left(\psi_{i-1,j}^{k}+\psi_{i+1,j}^{k}\right)+(\Delta x)^2\left(\psi_{i,j-1}^{k}+\psi_{i,j+1}^{k}\right)\right]}{2\hbar^2\left[(\Delta x)^2+(\Delta y)^2\right]+2V_{i,j}m(\Delta x)^2(\Delta y)^2}} \tag{22}$$

where $\psi_{i,j}^{k+1}$ is the wave function evaluated at the next iteration for the Jacobi method.

On the other hand, constructing ψ^{k+1} based on the latest approximations of the wave function, as suggested by the Gauss-Seidel method and considering the nodes (i+1) & (j+1) to be updated after the nodes (i) & (j), yields the resulting equation to be solved at every iteration:

$$\boxed{\psi_{i,j}^{k+1}=\frac{\hbar^2\left[(\Delta y)^2\left(\psi_{i-1,j}^{k}+\psi_{i+1,j}^{k}\right)+(\Delta x)^2\left(\psi_{i,j-1}^{k}+\psi_{i,j+1}^{k}\right)\right]}{2\hbar^2\left[(\Delta x)^2+(\Delta y)^2\right]+2V_{i,j}m(\Delta x)^2(\Delta y)^2}} \tag{23}$$

where $\psi_{i,j}^{k+1}$, $\psi_{i-1,j}^{k+1}$ and $\psi_{i,j+1}^{k+1}$ are the wave functions evaluated at the next iteration for the Gauss-Seidel method.

2.2. Time-dependent Schrödinger Equation (TDSE)

The time-dependent Schrödinger equation (TDSE), Eqn. (2), describes the evolution of a non-relativistic particle's wave function in a given potential energy field over time. It gives information about the time-dependent probability distribution of finding the particle in a given location and allows for studying quantum systems under various dynamic conditions, such as many-particle systems or time-dependent potentials [2, 3]. It can be mathematically represented as a *parabolic* partial differential equation (PDE), characterized by unique transient solutions for a given set of initial and boundary conditions [6].

TDSE plays a vital role in the understanding of various physical systems, particularly in quantum mechanics, quantum field theory, and quantum optics. For instance, in quantum mechanics, TDSE describes the evolution of a particle in a potential, whereas, in quantum optics, it describes the evolution of the electromagnetic field in a given cavity [2].

As with TISE, analytical solutions to TDSE are not always possible, particularly for complex systems and dynamic conditions. Therefore, numerical methods are used to approximate the solutions to TDSE. The numerical analysis of parabolic PDEs yields some methods with sufficiently accurate and stable solutions over time, making it possible to study the dynamic behavior of quantum systems [6].

2.2.1. One dimensional solution

In 1D, the position vector becomes $\mathbf{r} = (x)$ and TDSE, Eqn. (2), reduces to the following:

$$i\hbar \frac{\partial}{\partial t}\psi(x,t) = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2}\psi(x,t) + V(x)\psi(x,t) \tag{24}$$

In this project, to numerically evaluate the transient solution of Eqn. (24), explicit Euler and Crank-Nicolson methods are considered. Spatial discretization of Eqn. (24) using the finite differencing scheme given for 1D, see Eqn. (4), yields the following:

$$i\hbar \left(\frac{\partial \psi}{\partial t}\right)\bigg|_i = -\frac{\hbar^2}{2m}\left[\frac{\psi_{i-1} - 2\psi_i + \psi_{i+1}}{(\Delta x)^2}\right] + V_i\psi_i \tag{25}$$

According to the explicit Euler method, the temporal discretization of the time derivative of the wave function has the following form:

$$\left(\frac{\partial \psi}{\partial t}\right)\bigg|_i = \frac{\psi_i^{k+1} - \psi_i^k}{\Delta t} \tag{26}$$

where ψ_i^{k+1} is the wave function at the next time-step k, whereas the spatial part of Eqn. (25) is evaluated at the current time-step, k. The resulting equation is given as follows:

$$i\hbar\frac{\psi_i^{k+1} - \psi_i^k}{\Delta t} = -\frac{\hbar^2}{2m}\left[\frac{\psi_{i-1}^k - 2\psi_i^k + \psi_{i+1}^k}{(\Delta x)^2}\right] + V_i\psi_i^k \tag{27}$$

Rearranging Eqn. (27) to solve for ψ_i^{k+1} yields:

$$\boxed{\psi_i^{k+1} = \psi_i^k + \Delta t\left[-\frac{\hbar}{2im}\left[\frac{\psi_{i-1}^k - 2\psi_i^k + \psi_{i+1}^k}{(\Delta x)^2}\right] + \frac{V_i\psi_i^k}{i\hbar}\right]} \tag{28}$$

where ψ_i^{k+1} is the wave function evaluated at the next time-step for the explicit Euler method.

In contrast to the explicit Euler method, the Crank-Nicolson method is implicit and evaluates the spatial part of the wave function given in Eqn. (25) at $k + \frac{1}{2}$:

$$\begin{aligned}\left[-\frac{\hbar^2}{2m}\left[\frac{\psi_{i-1} - 2\psi_i + \psi_{i+1}}{(\Delta x)^2}\right] + V_i\psi_i\right]^{k+\frac{1}{2}} = \frac{1}{2}\Bigg[&-\frac{\hbar^2}{2m}\left[\frac{\psi_{i-1}^{k+1} - 2\psi_i^{k+1} + \psi_{i+1}^{k+1}}{(\Delta x)^2}\right] + V_i\psi_i^{k+1} + \\ &-\frac{\hbar^2}{2m}\left[\frac{\psi_{i-1}^k - 2\psi_i^k + \psi_{i+1}^k}{(\Delta x)^2}\right] + V_i\psi_i^k\Bigg]\end{aligned} \tag{29}$$

Noting the time discretization of the wave function according to the Crank-Nicolson method also has the form of Eqn. (26), combining Eqn. (25), Eqn. (26) and Eqn. (29) results in the following:

$$\begin{aligned}\left(-\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi_{i-1}^{\mathrm{k}+1}+\left(1+\frac{i\hbar\Delta t}{2m\Delta x^2}-\frac{V_i\Delta t}{2i\hbar}\right)\psi_i^{\mathrm{k}+1}+\left(-\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi_{i+1}^{\mathrm{k}+1}\\=\left(\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi_{i-1}^{\mathrm{k}}+\left(1-\frac{i\hbar\Delta t}{2m\Delta x^2}+\frac{V_i\Delta t}{2i\hbar}\right)\psi_i^{\mathrm{k}}+\left(\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi_{i+1}^{\mathrm{k}}\end{aligned} \tag{30}$$

Rearranging Eqn. (30) separating the terms to be evaluated at the next time-step, $k+1$, and the current time-step, k, yields:

$$\begin{aligned}\left(-\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi_{i-1}^{\mathrm{k}+1}+\left(1+\frac{i\hbar\Delta t}{2m\Delta x^2}-\frac{V_i\Delta t}{2i\hbar}\right)\psi_i^{\mathrm{k}+1}+\left(-\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi_{i+1}^{\mathrm{k}+1}\\=\left(\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi_{i-1}^{\mathrm{k}}+\left(1-\frac{i\hbar\Delta t}{2m\Delta x^2}+\frac{V_i\Delta t}{2i\hbar}\right)\psi_i^{\mathrm{k}}+\left(\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi_{i+1}^{\mathrm{k}}\end{aligned} \tag{31}$$

By denoting $c_1=\frac{i\hbar\Delta t}{4m\Delta x^2}$, $c_2=1+\frac{i\hbar\Delta t}{2m\Delta x^2}-\frac{V_i\Delta t}{2i\hbar}$, and $c_3=1+\frac{i\hbar\Delta t}{2m\Delta x^2}+\frac{V_i\Delta t}{2i\hbar}$ Eqn. (31) in the following simplified form:

$$\boxed{-c_1\psi_{i-1}^{\mathrm{k}+1}+c_2\psi_i^{\mathrm{k}+1}-c_1\psi_{i+1}^{\mathrm{k}+1}=c_1\psi_{i-1}^{\mathrm{k}}+c_3\psi_i^{\mathrm{k}}+c_1\psi_{i+1}^{\mathrm{k}}} \tag{32}$$

where ψ_{i-1}^{k+1}, ψ_i^{k+1}, and $\psi_{i+1}^{s}k+1$ are the wave functions evaluated at the next time-step for the Crank-Nicolson method.

Eqn. (32) can further be written as a tri-diagonal matrix, where the nodes (i-1), (i), and (i+1) construct the lower diagonal, main diagonal, and upper diagonal elements, respectively. The resulting equation having a form $\mathbf{A}\boldsymbol{\psi}^{k+1}=\mathbf{B}\boldsymbol{\psi}^{k}=\mathbf{c}^{k}$ where $\mathbf{A}$ and $\mathbf{B}$ are matrices consisting elements of c_1, c_2, c_3, and $\boldsymbol{\psi}$ is the wave function vector, can then be solved using direct methods such as Gauss elimination or iterative methods such as Thomas algorithm to achieve arithmetically exact solutions. Due to the limited number of digits, even the exact solutions are prone to round-off errors, and the error may accumulate during time iterations.

2.2.2. Two dimensional solution

In 2D, the position vector becomes $\mathbf{r}=(x,y)$ and TDSE, Eqn. (2), reduces to the following:

$$i\hbar\frac{\partial}{\partial t}\psi(x,y,t)=-\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)\psi(x,y,t)+V(x,y)\psi(x,y,t) \tag{33}$$

To numerically evaluate the transient solution of Eqn. (33) with the spatial discretization of Eqn. (33) using the finite differencing scheme given for 2D, see Eqn. (5), yields the following:

$$i\hbar\left(\frac{\partial\psi}{\partial t}\right)\bigg|_{i,j}=-\frac{\hbar^2}{2m}\left[\frac{\psi_{i-1,j}-2\psi_{i,j}+\psi_{i+1,j}}{(\Delta x)^2}+\frac{\psi_{i,j-1}-2\psi_{i,j}+\psi_{i,j+1}}{(\Delta y)^2}\right]+V_{i,j}\psi_{i,j} \tag{34}$$

According to the explicit Euler method, the temporal discretization of the time derivative of the wave function has the following form:

$$\left(\frac{\partial\psi}{\partial t}\right)\bigg|_{i,j}=\frac{\psi_{i,j}^{k+1}-\psi_{i,j}^{k}}{\Delta t} \tag{35}$$

where $\psi_{i,j}^{k+1}$ is the wave function at the next time-step k, whereas the spatial part of Eqn. (34) is evaluated at the current time-step, k. The resulting equation is given as follows:

$$\frac{\psi_{i,j}^{k+1}-\psi_{i,j}^{k}}{\Delta t}=-\frac{\hbar^2}{2m}\left[\frac{\psi_{i-1,j}-2\psi_{i,j}+\psi_{i+1,j}}{(\Delta x)^2}+\frac{\psi_{i,j-1}-2\psi_{i,j}+\psi_{i,j+1}}{(\Delta y)^2}\right]+V_{i,j}\psi_{i,j} \tag{36}$$

Rearranging Eqn. (36) to solve for $\psi_{i,j}^{k+1}$ yields:

$$\boxed{\psi_{i,j}^{k+1}=\psi_{i,j}^{k}+\Delta t\left[-\frac{\hbar}{2im}\left[\frac{\psi_{i-1,j}^{k}-2\psi_{i,j}^{k}+\psi_{i+1,j}^{k}}{(\Delta x)^2}+\frac{\psi_{i,j-1}^{k}-2\psi_{i,j}^{k}+\psi_{i,j+1}^{k}}{(\Delta y)^2}\right]+\frac{V_{i,j}\psi_{i}^{k}}{i\hbar}\right]} \tag{37}$$

where ψ_{i}^{k+1} is the wave function evaluated at the next time-step for the explicit Euler method.

For the Crank-Nicolson method, the spatial part of the wave function that is given in Eqn. (33) can be evaluated at $k+\frac{1}{2}$:

$$\begin{aligned}&i\hbar\left[-\frac{\hbar^2}{2m}\left[\frac{\psi_{i-1,j}-2\psi_{i,j}+\psi_{i+1,j}}{(\Delta x)^2}+\frac{\psi_{i,j-1}-2\psi_{i,j}+\psi_{i,j+1}}{(\Delta y)^2}\right]+V_{i,j}\psi_{i,j}\right]^{k+\frac{1}{2}}\\&=\frac{1}{2}\left[-\frac{\hbar^2}{2m}\left[\frac{\psi_{i-1,j}^{k+1}-2\psi_{i,j}^{k+1}+\psi_{i+1,j}^{k+1}}{(\Delta x)^2}+\frac{\psi_{i,j-1}^{k+1}-2\psi_{i,j}^{k+1}+\psi_{i,j+1}^{k+1}}{(\Delta y)^2}\right]+V_{i,j}\psi_{i,j}^{k+1}+\right.\\&\qquad\left.-\frac{\hbar^2}{2m}\left[\frac{\psi_{i-1,j}^{k}-2\psi_{i,j}^{k}+\psi_{i+1,j}^{k}}{(\Delta x)^2}+\frac{\psi_{i,j-1}^{k}-2\psi_{i,j}^{k}+\psi_{i,j+1}^{k}}{(\Delta y)^2}\right]+V_{i,j}\psi_{i,j}^{k}\right]\end{aligned} \tag{38}$$

Noting the time discretization of the wave function according to the Crank-Nicolson method also has the form of Eqn. (35), combining Eqn. (34), Eqn. (35) and Eqn. (38) results in the following:

$$\begin{aligned}\left(\frac{\partial\psi}{\partial t}\right)\Bigg|_{i,j}&=\frac{1}{2}\left[-\frac{\hbar^2}{2m}\left[\frac{\psi_{i-1,j}^{k+1}-2\psi_{i,j}^{k+1}+\psi_{i+1,j}^{k+1}}{(\Delta x)^2}+\frac{\psi_{i,j-1}^{k+1}-2\psi_{i,j}^{k+1}+\psi_{i,j+1}^{k+1}}{(\Delta y)^2}\right]+V_{i,j}\psi_{i,j}^{k+1}+\right.\\&\qquad\left.-\frac{\hbar^2}{2m}\left[\frac{\psi_{i-1,j}^{k}-2\psi_{i,j}^{k}+\psi_{i+1,j}^{k}}{(\Delta x)^2}+\frac{\psi_{i,j-1}^{k}-2\psi_{i,j}^{k}+\psi_{i,j+1}^{k}}{(\Delta y)^2}\right]+V_{i,j}\psi_{i,j}^{k}\right]\end{aligned} \tag{39}$$

Rearranging Eqn. (39) separating the terms to be evaluated at the next time-step, $k+1$, and the current time-step, k, yields:

$$\begin{aligned}&\left(-\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi_{i-1,j}^{k+1}+\left(-\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi_{i+1,j}^{k+1}+\left(1+\frac{i\hbar\Delta t}{2m\Delta x^2}+\frac{i\hbar\Delta t}{2m\Delta y^2}-\frac{V_{i,j}\Delta t}{2i\hbar}\right)\psi_{i,j}^{k+1}\\&+\left(-\frac{i\hbar\Delta t}{4m\Delta y^2}\right)\psi_{i,j-1}^{k+1}+\left(-\frac{i\hbar\Delta t}{4m\Delta y^2}\right)\psi_{i,j+1}^{k+1}=\left(-\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi_{i-1,j}^{k}+\left(-\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi_{i+1,j}^{k}\\&+\left(1-\frac{i\hbar\Delta t}{2m\Delta x^2}-\frac{i\hbar\Delta t}{2m\Delta y^2}+\frac{V_{i,j}\Delta t}{2i\hbar}\right)\psi_{i,j}^{k}+\left(-\frac{i\hbar\Delta t}{4m\Delta y^2}\right)\psi_{i,j-1}^{k}+\left(-\frac{i\hbar\Delta t}{4m\Delta y^2}\right)\psi_{i,j+1}^{k}\end{aligned} \tag{40}$$

By denoting $d_1 = \frac{i\hbar\Delta t}{4m\Delta x^2}$, $d_2 = 1 + \frac{i\hbar\Delta t}{2m\Delta x^2} + \frac{i\hbar\Delta t}{2m\Delta y^2} - \frac{V_{i,j}\Delta t}{2i\hbar}$, $d_3 = \frac{i\hbar\Delta t}{4m\Delta y^2}$ and $d_4 = 1 - \frac{i\hbar\Delta t}{2m\Delta x^2} - \frac{i\hbar\Delta t}{2m\Delta y^2} + \frac{V_{i,j}\Delta t}{2i\hbar}$, Eqn. (40) can be expressed in the following simplified form:

$$\boxed{-d_1\psi^{k+1}_{i-1,j} - d_1\psi^{k+1}_{i+1,j} + d_2\psi_{i,}{}_j{}^{k+1} - d_3\psi^{k+1}_{i,j-1} - d_3\psi^{k+1}_{i,j+1} = d_1\psi^{k}_{i-1,j} + d_1\psi^{k}_{i+1,j} + d_4\psi_{i,}{}_j{}^{k} + d_3\psi^{k}_{i,j-1} + d_3\psi^{k}_{i,j+1}} \tag{41}$$

where $\psi^{k+1}_{i-1,j}$, $\psi^{k+1}_{i,j-1}$, $\psi^{k+1}_{i,j}$, $\psi^{k+1}_{i,j-1}$, and $\psi^{k+1}_{i,j+1}$ are the wave functions evaluated at the next time-step for the Crank-Nicolson method.

However, the caveat of Eqn. (41) is such that it cannot be written as a tri-diagonal matrix like Eqn. (32), but rather as a penta-diagonal matrix. Although there exist algorithms for solving penta-diagonal systems resembling Thomas algorithm, for the numerical evaluation of Eqn. (32), alternating direction implicit method (ADI) is considered in the context of this project.

The alternating direction implicit (ADI) method divides the time discretization into two steps; at the first step, it considers the x-dimension of the spatial discretization implicit by keeping the y-dimension explicit, and at the second step, it considers the y-dimension implicit by using the evaluated values of x-dimension at the first step. At each step, the spatial discretization is obtained using the Crank-Nicolson method, and a total of one time-step is evaluated at the end of the second ADI step.

According to ADI, Eqn. (40) can be separated into two equations:

$$\begin{aligned}\left(-\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi^{k+\frac{1}{2}}_{i-1,j} + \left(1 + \frac{i\hbar\Delta t}{2m\Delta x^2} - \frac{V_{i,j}\Delta t}{4i\hbar}\right)\psi^{k+\frac{1}{2}}_{i,j} + \left(-\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi^{k+\frac{1}{2}}_{i+1,j} \\ = \left(\frac{i\hbar\Delta t}{4m\Delta y^2}\right)\psi^{k}_{i,j-1} + \left(1 - \frac{i\hbar\Delta t}{2m\Delta y^2} + \frac{V_{i,j}\Delta t}{4i\hbar}\right)\psi^{k}_{i,j} + \left(\frac{i\hbar\Delta t}{4m\Delta y^2}\right)\psi^{k}_{i,j+1}\end{aligned} \tag{42}$$

$$\begin{aligned}\left(-\frac{i\hbar\Delta t}{4m\Delta y^2}\right)\psi^{k+1}_{i,j-1} + \left(1 + \frac{i\hbar\Delta t}{2m\Delta y^2} - \frac{V_{i,j}\Delta t}{4i\hbar}\right)\psi^{k+1}_{i,j} + \left(-\frac{i\hbar\Delta t}{4m\Delta y^2}\right)\psi^{k+1}_{i,j+1} \\ = \left(\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi^{k+\frac{1}{2}}_{i-1,j} + \left(1 - \frac{i\hbar\Delta t}{2m\Delta x^2} + \frac{V_{i,j}\Delta t}{4i\hbar}\right)\psi^{k+\frac{1}{2}}_{i,j} + \left(\frac{i\hbar\Delta t}{4m\Delta x^2}\right)\psi^{k+\frac{1}{2}}_{i+1,j}\end{aligned} \tag{43}$$

By denoting $e_1 = \frac{i\hbar\Delta t}{4m\Delta x^2}$, $e_2 = 1 + \frac{i\hbar\Delta t}{2m\Delta x^2} - \frac{V_{i,j}\Delta t}{4i\hbar}$, $e_3 = \frac{i\hbar\Delta t}{4m\Delta y^2}$, $e_4 = 1 - \frac{i\hbar\Delta t}{2m\Delta y^2} + \frac{V_{i,j}\Delta t}{4i\hbar}$, $e_5 = 1 + \frac{i\hbar\Delta t}{2m\Delta y^2} - \frac{V_{i,j}\Delta t}{4i\hbar}$ and $e_6 = 1 - \frac{i\hbar\Delta t}{2m\Delta x^2} + \frac{V_{i,j}\Delta t}{4i\hbar}$, Eqn. (42) and Eqn. (43) can be expressed in the following simplified forms:

$$\boxed{-e_1\psi^{k+\frac{1}{2}}_{i-1,j} + e_2\psi^{k+\frac{1}{2}}_{i,j} - e_1\psi^{k+\frac{1}{2}}_{i+1,j} = e_3\psi^{k}_{i,j-1} + e_4\psi^{k}_{i,j} + e_3\psi^{k}_{i,j+1}} \tag{44}$$

$$\boxed{-e_3\psi^{k+1}_{i,j-1} + e_5\psi^{k+1}_{i,j} - e_3\psi^{k+1}_{i,j+1} = e_1\psi^{k+\frac{1}{2}}_{i-1,j} + e_6\psi^{k+\frac{1}{2}}_{i,j} + e_1\psi^{k+\frac{1}{2}}_{i+1,j}} \tag{45}$$

where $\psi^{k+\frac{1}{2}}_{i-1,j}$, $\psi^{k+\frac{1}{2}}_{i,j}$, and $\psi^{k+\frac{1}{2}}_{i+1,j}$ are the wave functions evaluated at the first step of ADI; and $\psi^{k+1}_{i,j-1}$, $\psi^{k+1}_{i,j}$, and $\psi^{k+1}_{i,j+1}$ are the wave functions evaluated at the second step of ADI.

With this, Eqn. (44) and Eqn. (45) can further be written as with tri-diagonal matrices, where the nodes (i-1) & (j-1), (i) & (j), and (i+1) & (j+1) construct the lower diagonal, main diagonal, and upper diagonal elements,

respectively. The resulting equations having the forms $\mathbf{A_1}\boldsymbol{\psi}^{k+\frac{1}{2}} = \mathbf{B_1}\boldsymbol{\psi}^{k} = \mathbf{c_1}^{k}$ and $\mathbf{A_2}\boldsymbol{\psi}^{k+1} = \mathbf{B_2}\boldsymbol{\psi}^{k+\frac{1}{2}} = \mathbf{c_2}^{k+\frac{1}{2}}$ where $\mathbf{A_1}$, $\mathbf{A_2}$, $\mathbf{B_1}$ and $\mathbf{B_2}$ are matrices consisting elements of e_1, e_2, e_3, e_4, e_5, e_6, and $\boldsymbol{\psi}$ is the wave function matrix, can then be solved using direct methods such as Gauss elimination or iterative methods such as Thomas algorithm to achieve arithmetically exact solutions. Due to the two-step approach of ADI, the Gauss elimination or Thomas algorithm must be applied twice, once at each ADI step.

3. Results and Discussion

3.1. Time-independent Schrödinger Equation (TISE)

According to the methods presented in Section: Numerical Modeling, the numerical solution of TISE for one and two-dimensional spatial domains are represented in the following sections.

3.1.1. One dimensional solution

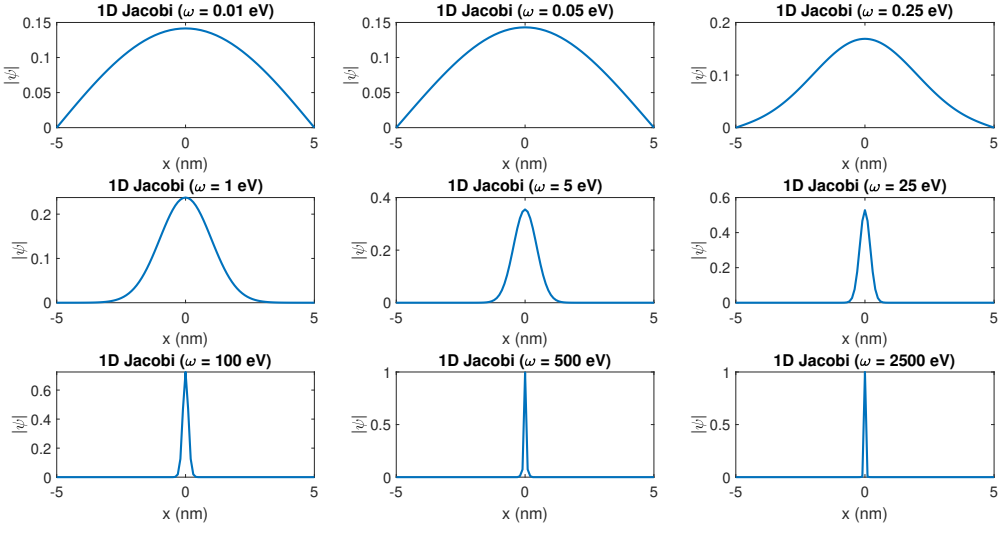

Figure 2: Solution of 1D time-independent Schrödinger Equation using Jacobi at different frequencies

For the one-dimensional analysis, the time-independent Schrödinger equation given in Eqn. (24), is considered with a quantum harmonic oscillator potential $V^* = \frac{1}{2}m\omega^2x^2$, m is the mass of the particle, ω is the angular frequency of the oscillator.

When the oscillator's frequency is increased, the potential energy barrier experienced by the particle becomes steeper. The wave function, which describes the probability density of finding the particle at a specific position, will be more localized as the particle's motion is more constrained by the steeper potential energy barrier.

With the increase in frequency, the system's energy increases; hence the energy eigenvalues will also increase.

And it is visible that the wave function will be more localized as the particle's motion is more constrained by the steeper potential energy barrier. As a result, the particle will have a higher probability of being found near the minimum of the potential energy well, which is at $x = 0$ for the harmonic oscillator.

Additionally, one could also note that the wave function would be more localized in space as well, and also the energy spacing between successive energy levels will also increase, this implies that the transitions between these energy levels will be less probable and hence energy absorption/emission will also be less probable.

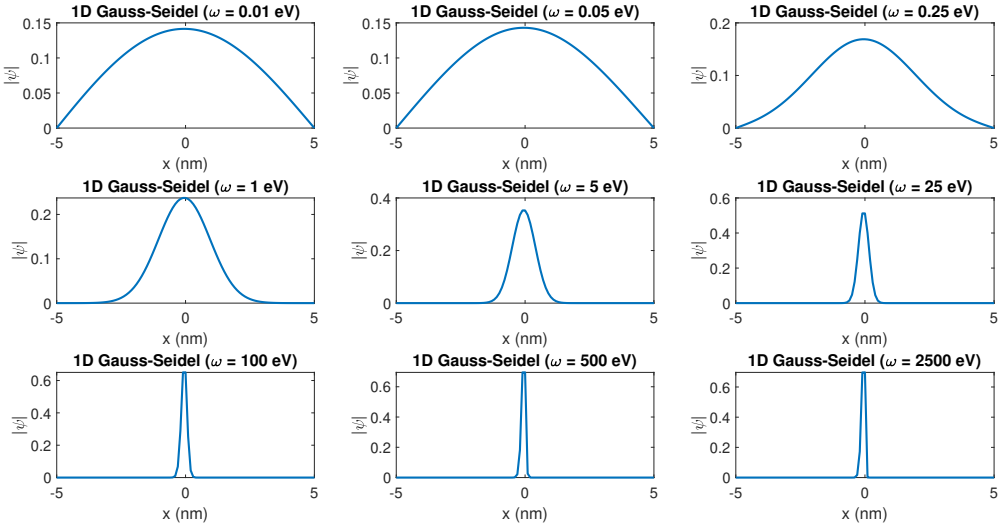

Figure 3: Solution of 1D time-independent Schrödinger equation using Gauss-Seidel at different frequencies

The one-dimensional time-independent Schrödinger equation for a harmonic oscillator can also be solved using the Gauss-Seidel method. This method involves iteratively solving for the wave function at each point in space, using the wave function values at nearby points as input.

The wave function will also change when the oscillator's frequency increases due to the steeper potential energy barrier. The wave function will become more tightly confined around the minimum of the potential energy well, which is at x = 0. As the Gauss-Seidel method is based on an iterative process, these changes in the wave function will be reflected in the convergence of the solution.

One of the key features of the Gauss-Seidel method is that it ensures the solution of the wave function at each iteration point is taken into account in the next iteration, which results in faster convergence. This differs from the Jacobi method, where the wave function at each iteration point is not used until the next iteration.

Additionally, with the increase in frequency, the energy eigenvalues will also increase, resulting in a more localized wave function; thus, the number of iterations required for convergence will decrease.

In conclusion, when using the Gauss-Seidel method to solve the Schrödinger equation for a harmonic oscillator

with an increased frequency, the wave function will become more tightly confined around the minimum of the potential energy well, and the number of iterations needed for convergence will decrease. This method is also efficient as it considers the wave function at each iteration point which speeds up the convergence process.

3.1.2. Two dimensional solution

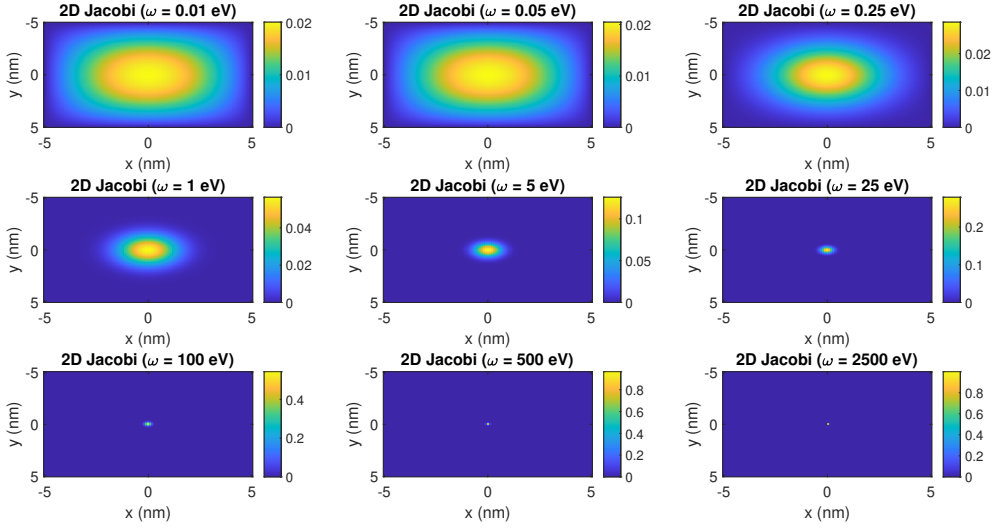

Figure 4: Solution of 2D time-independent Schrödinger Equation using Jacobi at different frequencies

For the one-dimensional analysis, the time-independent Schrödinger equation given in Eqn. (33), is considered with a quantum harmonic oscillator potential $V^* = \frac{1}{2}m\omega^2(x^2+y^2)$, m is the mass of the particle, ω is the angular frequency of the oscillator.

If the two-dimensional time-independent Schrödinger equation for a harmonic oscillator is solved using the Jacobi method, and the absolute value of the wave functions is plotted at nine different frequencies in increasing order, the plots are likely to show localization of the wave function as the frequency increases.

At lower frequencies, the wave function will be relatively spread out and have a relatively large amplitude across a wide range of positions in the x-y plane. As the frequency increases, the wave function will become more localized, and its amplitude will become smaller in most regions of the x-y plane. The wave function will be highly localized at the highest frequency, with most of its amplitude concentrated in a small region near the origin.

Additionally, one can notice that the shapes of the plots will change with the increase in frequency; at lower frequencies, the wave function will be relatively smooth and symmetric; as frequency increases, the wave function will become more complex, and the symmetry will be broken. It will have a more complex pattern and will be less smooth.

It is also important to note that, as the frequency increases, the energy eigenvalues also increase; hence the wave

function will be associated with a higher energy state.

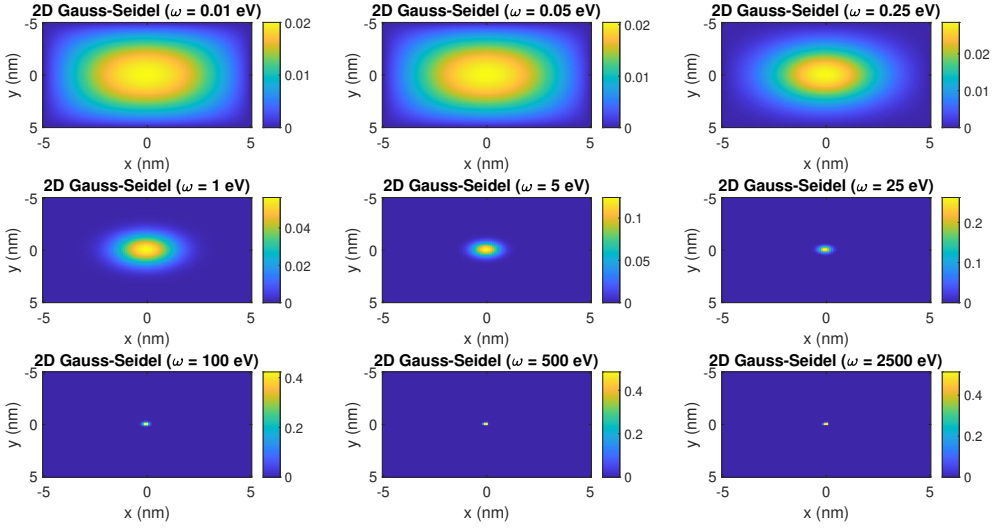

Figure 5: Solution of 2D time-independent Schrödinger Equation using Gauss-Seidel at different frequencies

If the two-dimensional time-independent Schrödinger equation for a harmonic oscillator is solved using the Gauss-Seidel method, and the absolute value of the wave functions is plotted at nine different frequencies in increasing order, the plots are likely to show a similar localization of the wave function as the frequency increases, as in the plots obtained using Jacobi method.

As in the case of the Jacobi method, as the frequency increases, the wave function becomes more concentrated, and its amplitude decreases in most regions of the x-y plane. At the highest frequency, the wave function will be highly concentrated, with most of its amplitude located in a small area close to the origin.

One of the key features of the Gauss-Seidel method is that it takes into account the wave function at each iteration point which speeds up the convergence process; this method is beneficial for solving large systems of equations. While the Gauss-Seidel method is different from the Jacobi method in terms of implementation, the final solution obtained from both methods will be similar.

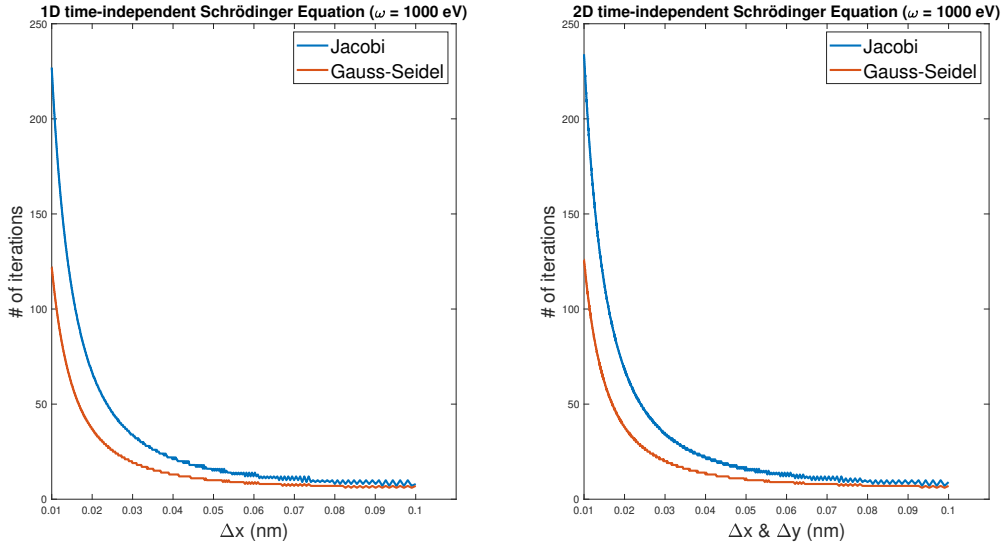

Figure 6: Number of iterations for Jacobi and Gauss-Seidel to reach convergence regarding to the change in the size of the grid

Jacobi and Gauss-Seidel are both iterative methods to solve partial differential equations, such as the time-independent Schrödinger equation. Both methods can be used to find approximate solutions to large, sparse systems of linear equations, but they differ in how the solution is updated at each iteration.

In the Jacobi method, the solution is updated by using the values of the solution from the previous iteration to estimate the values of the solution at the current iteration. This process is repeated until the solution converges to a fixed point. One of the main advantages of Jacobi is that it is simple and easy to implement, but it is relatively slow. Also, it can only be used for a diagonally dominant or a positive definite matrix.

On the other hand, the Gauss-Seidel method updates the solution by using the values of the solution from the current iteration to estimate the values of the solution at the next iteration. This process is also repeated until the solution converges to a fixed point. The Gauss-Seidel method is generally more efficient than the Jacobi method because it uses the updated values of the solution at each iteration. However, it also can only be used for a diagonally dominant matrix.

Regarding the number of iterations required to solve the time-independent Schrödinger equation, both Jacobi and Gauss-Seidel will require more iterations as the grid size increases. This is because as the grid size increases, the problem becomes more complex and, therefore, more difficult to solve. However, the Gauss-Seidel method is generally more efficient than the Jacobi method, so it will typically require fewer iterations to reach convergence.

In the 1D time-independent Schrödinger equation, the Jacobi method requires a higher number of iterations than the Gauss-Seidel to reach convergence. But this may not be the case for the 2D time-independent Schrödinger equation as it largely depends on the problem at hand and the initial guess for the solution.

In summary, Jacobi and Gauss-Seidel are both iterative methods to solve partial differential equations. The main difference between them is how the solution is updated at each iteration. The Jacobi method is simple to implement but relatively slow, and the Gauss-Seidel method is more efficient but can only be used for a diagonally dominant matrix. The number of iterations required to solve the time-independent Schrödinger equation will increase as the grid size increases, but the Gauss-Seidel method will typically require fewer iterations than the Jacobi method to reach convergence.

3.2. Time-dependent Schrödinger Equation (TDSE)

According to the methods presented in Section: Numerical Modeling, the numerical solution of TDSE for one and two-dimensional spatial domains are represented in the following sections.

3.2.1. One dimensional solution

For one dimensional solution of TDSE, the explicit Euler method requires Eqn. (28) to be evaluated at every time step. For $V^* = 0$, see Eqn. (6), the free particle inside an infinite potential well solution is obtained for different time steps and presented as follows:

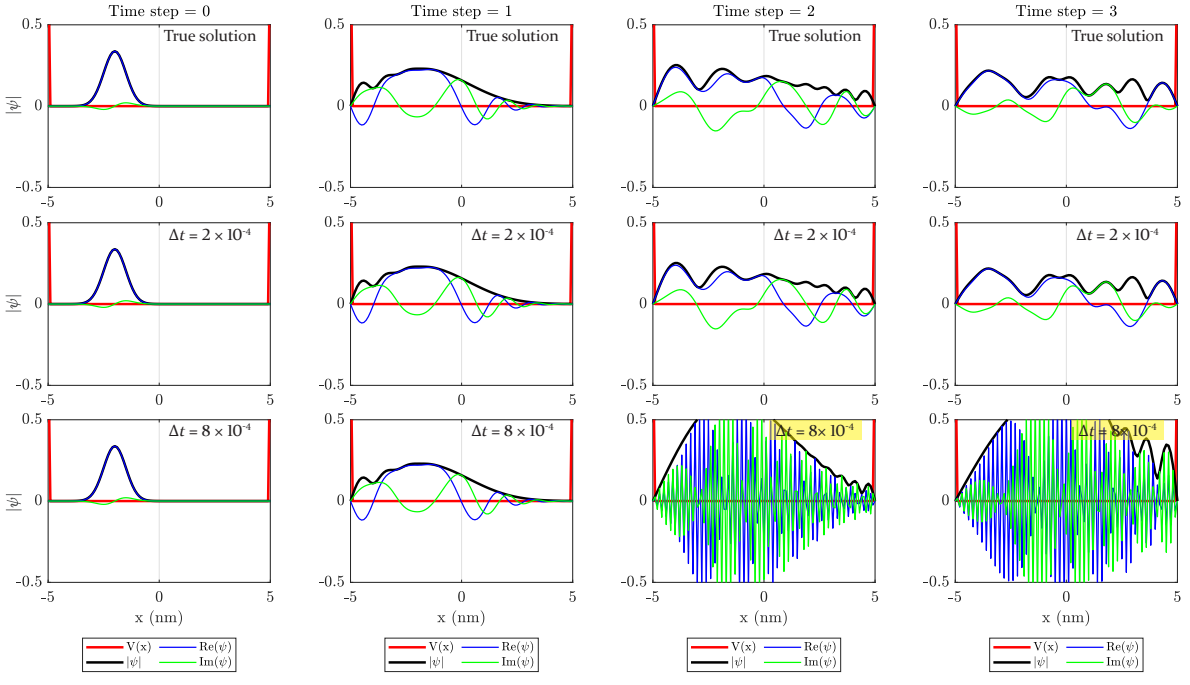

Figure 7: Snapshots of a free particle in an infinite potential using the explicit Euler method in 1D. Although the simulation is stable for $\Delta = 2 \times 10^{-4}$, $\Delta = 8 \times 10^{-4}$ results in an unstable simulation after accumulating errors.

The error is calculated by the absolute value of the deviation of the calculated wave function and true wave

function, integrated over the computational domain, $|\int(\psi - \psi_{true})dx|$. Since ψ_{true} is analytically unknown, $\Delta t = 2 \times 10^{-5}$ is assumed to represent the true solution. The resultant error is found as follows:

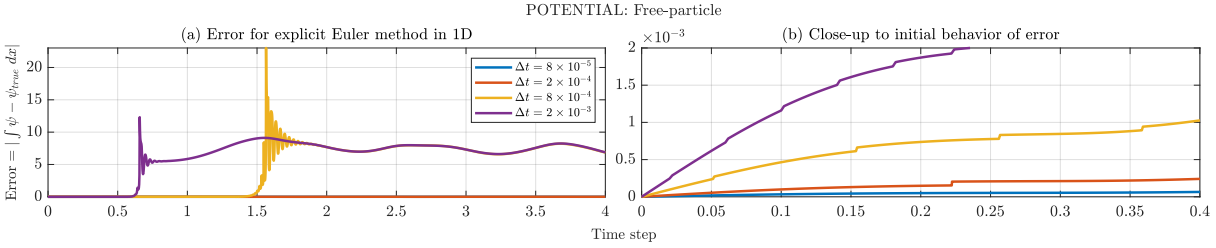

Figure 8: Error for the explicit Euler solutions in 1D, evaluated for different time steps (Δt). Accumulation of error, see part (b), represents O(Δt) dependence of error for the explicit Euler method. The sudden increases of the error observed initially for $\Delta t = 2 \times 10^{-3}$ and then for $\Delta t = 8 \times 10^{-4}$ shows the time steps where the numerical solution becomes unstable. The divergence of the solutions is prevented due to the normalization of the wave functions, see 10.

On the other hand, the Crank-Nicolson method requires Eqn. (32) to be evaluated at every time step. Similarly, for $V^* = 0$, see Eqn. (6), the free particle inside an infinite potential well solution is obtained for different time steps and presented as follows:

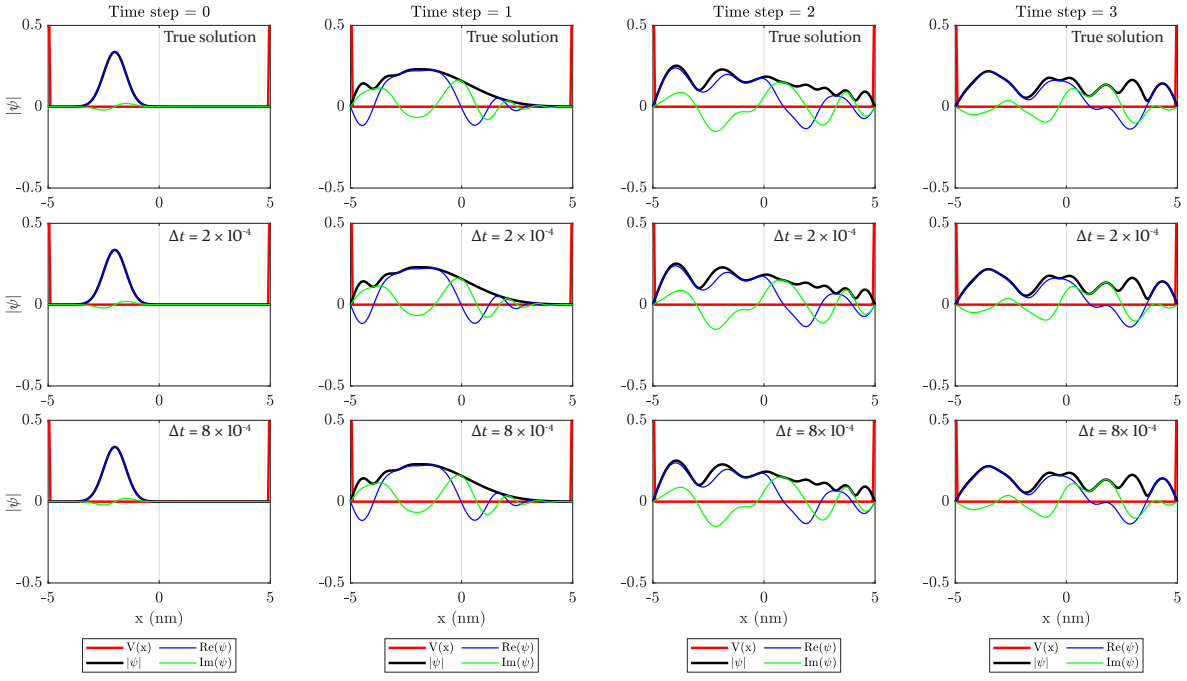

Figure 9: Snapshots of a free particle in an infinite potential using the Crank-Nicolson method in 1D. Both simulations are stable for Δt of choice.

The error is again calculated by the absolute value of the deviation of the calculated wave function and true wave function, integrated over the computational domain, $|\int \psi - \psi_{true} dx|$. The resultant error is found as follows:

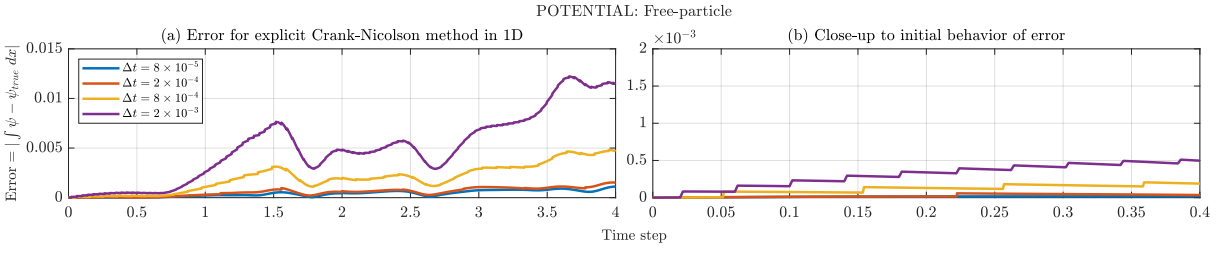

Figure 10: Error for the Crank-Nicolson solutions in 1D, evaluated for different time steps (Δt). Accumulation of error, see part (b), represents $O(\Delta t^2)$ dependence of error for the Crank-Nicolson method. As the Crank-Nicolson methods are unconditionally unstable, the accumulated error does not result in unstable solutions as it does for the explicit Euler method, see Fig. (8).

Lastly, the quantum tunneling phenomena may be demonstrated with a finite-well potential. A comparison between the true solution, and explicit Euler and Crank-Nicolson solutions evaluated at the identical time step

($\Delta t = 8 \times 10^{-4}$) is represented as follows:

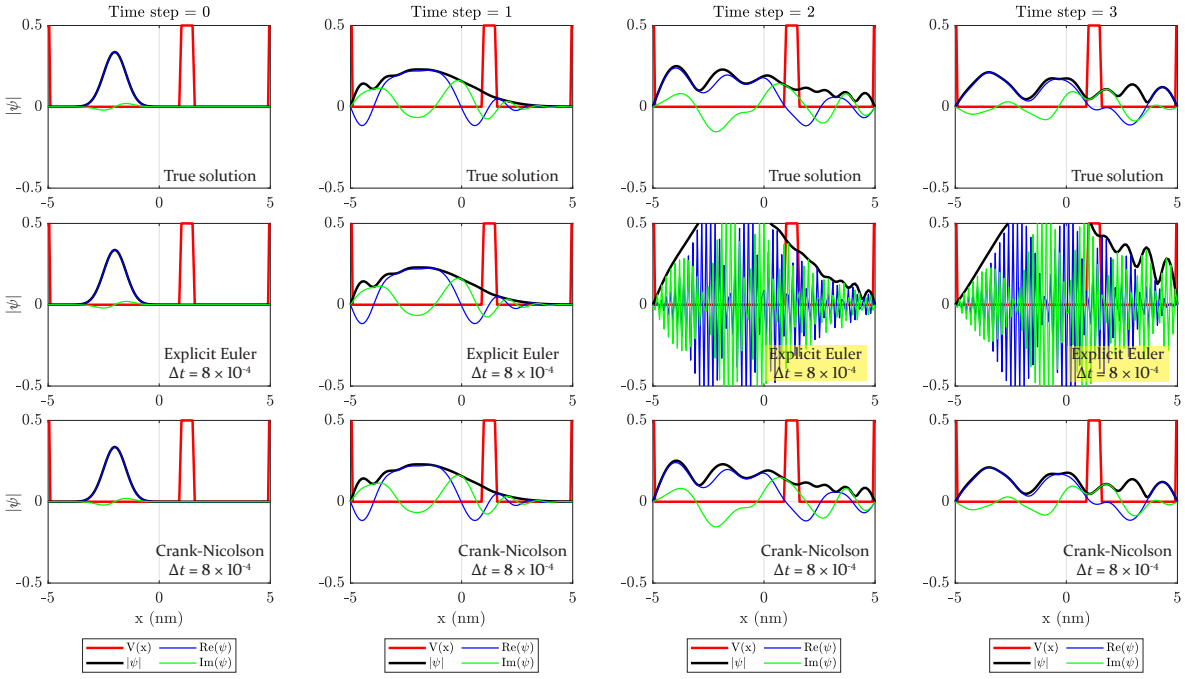

Figure 11: Numerical modeling of the quantum tunneling phenomena by trapping a particle within a finite-potential. For $\Delta t = 8 \times 10^{-4}$, the Crank-Nicolson method provides a stable solution, whereas the explicit Euler method results in an unsteady solution after accumulated errors.

3.2.2. Two dimensional solution

For two dimensional solution of TDSE, the explicit Euler method requires Eqn. (37) to be evaluated at every time step. For $V^* = 0$ over the spatial domain, the free particle inside an infinite potential well solution is obtained for different time steps and presented as follows:

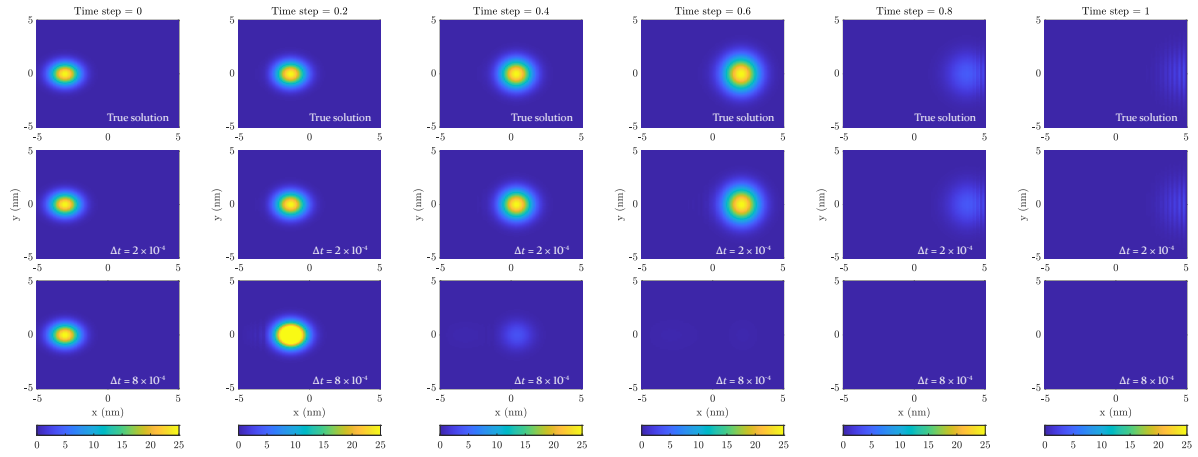

Figure 12: Snapshots of a free particle in an infinite potential using the explicit Euler method in 2D. Although the simulation is stable for $\Delta = 2 \times 10^{-4}$, $\Delta = 8 \times 10^{-4}$ results in an unstable simulation after accumulating errors.

The error is calculated by the absolute value of the deviation of the calculated wave function and true wave function, integrated over the computational domain, $|\int\int \psi - \psi_{true} dxdy|$. Since ψ_{true} is analytically unknown, $\Delta t = 2 \times 10^{-5}$ is assumed to represent the true solution. The resultant error is found as follows:

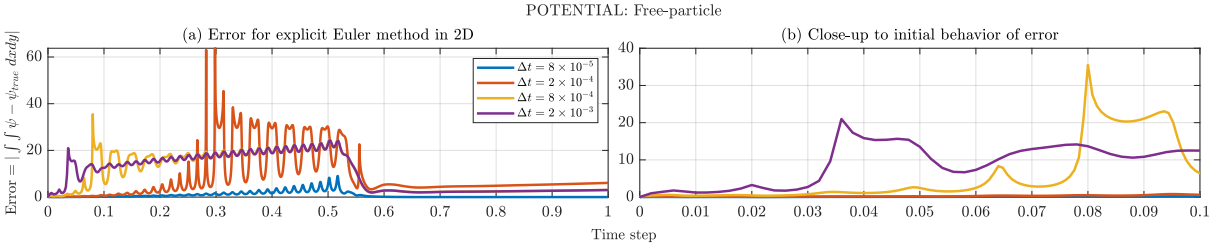

Figure 13: Error for the explicit Euler solutions in 2D, evaluated for different time steps (Δt). Accumulation of error, see part (b), represents O(Δt) dependence of error for the explicit Euler method. The sudden increases of the error observed initially for $\Delta t = 2 \times 10^{-3}$ and then for $\Delta t = 8 \times 10^{-4}$ shows the time steps where the numerical solution becomes unstable. Similarly, the divergence of the solutions is prevented due to the normalization of the wave functions, see 10.

On the other hand, the Crank-Nicolson method with ADI requires Eqn. (44) and Eqn. (45) to be evaluated iteratively at every time step. Similarly, for $V^* = 0$, the free particle inside an infinite potential well solution is obtained for different time steps and presented as follows:

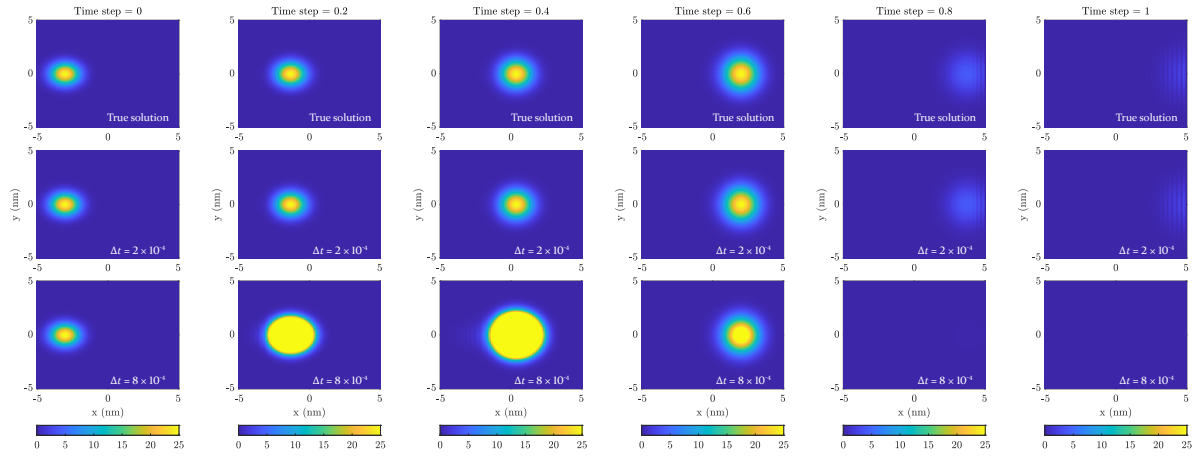

Figure 14: Snapshots of a free particle in an infinite potential using the Crank-Nicolson method in 2D. Both simulations are stable for Δt of choice.

The error is again calculated by the absolute value of the deviation of the calculated wave function and true wave function, integrated over the computational domain, $|\int\int \psi - \psi_{true} dxdy|$. The resultant error is found as follows:

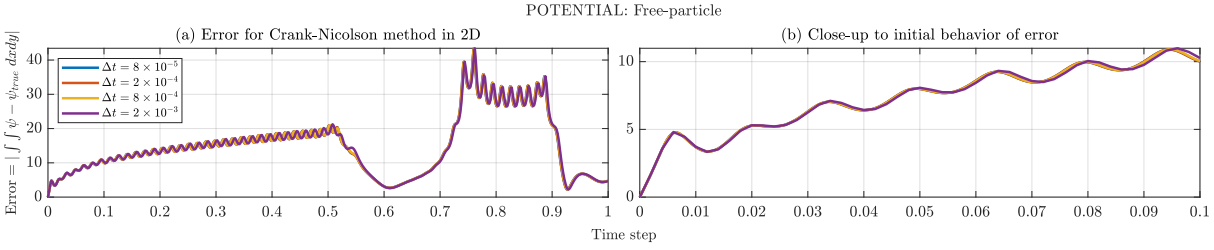

Figure 15: Error for the Crank-Nicolson solutions in 2D, evaluated for different time steps (Δt). Accumulation of error, see part (b), represents $O(\Delta t^2)$ dependence of error for the Crank-Nicolson method. As the Crank-Nicolson methods are unconditionally unstable, the accumulated error does not result in unstable solutions as it does for the explicit Euler method, see Fig. (13).

Lastly, the single-slit and double-slit experiments with specially placed finite-well potentials may be demonstrated. The modeling is only done using the explicit Euler method with time step ($\Delta t = 2 \times 10^{-4}$).:

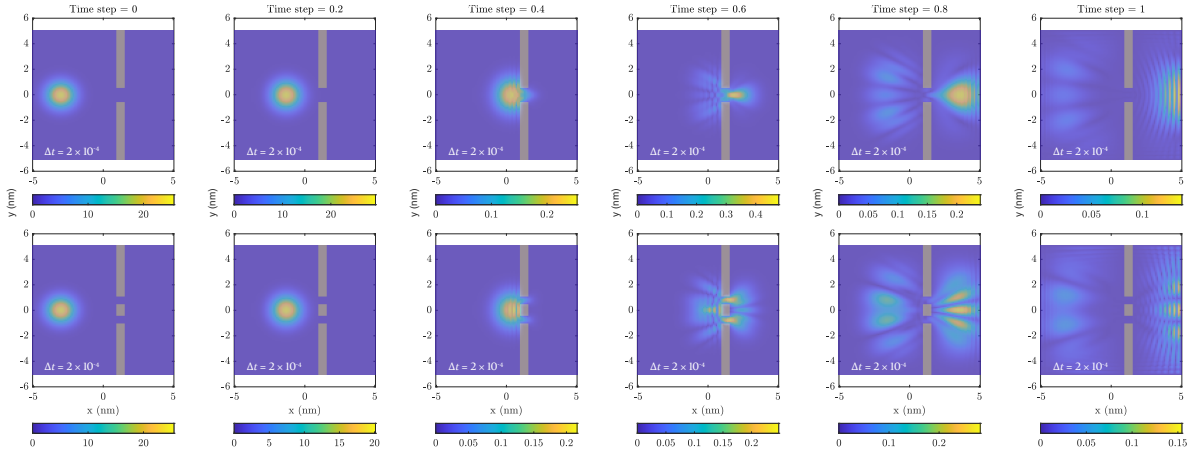

Figure 16: Numerical modeling of well-known single-slit and double-slit experiments by trapping a particle within a finite-potential. The modeling uses the explicit Euler with $\Delta t = 2 \times 10^{-5}$, yielding stable solutions within the time interval of interest.

4. Conclusions

This project has comprehensively investigated various numerical methods for solving the time-independent and time-dependent Schrödinger equation. The project has successfully solved the time-independent Schrödinger equation for one-dimensional and two-dimensional quantum systems by utilizing Jacobi and Gauss-Seidel iterative methods. Additionally, the explicit Euler and Crank-Nicolson methods (with Alternating Direction Implicit method for 2D) have been implemented to tackle the time-dependent Schrödinger equation for one-dimensional and two-dimensional quantum systems respectively.

For the time-dependent Schrödinger equation, the results of this project were evaluated by comparing the numerical solutions with the true solutions, where the numerical solutions with smaller time steps were approximated as the true solutions. It was found that the error of the numerical solutions shows the expected dependence to Δt, while solutions with comparable small Δt provide a high degree of accuracy. Although the system size was limited with $101/101 \times 101$ grids for one-dimensional/two-dimensional solutions, the solutions were time-efficient as the total runtime of any simulation did not exceed an hour. However, for smaller time steps, the wave function stored for each Δt became inefficient in terms of memory as the generated 3D matrices, including two spatial dimensions and single time dimension, occupied 1-2 GBs of RAM. As the Crank-Nicolson method with ADI was shown to be unconditionally stable, this amount can be significantly reduced compared to the explicit Euler method solutions. With this, it can be concluded that this project suggests that the numerical methods used in the project are efficient and reliable ways of solving the Schrödinger equation.

Lastly, the project has highlighted the importance of numerical methods in quantum mechanics. In particular,

it has shown that the Schrödinger equation, which is the foundation of quantum mechanics, can be tackled by numerical approaches. This opens up opportunities for further research in this area, such as investigating the behavior of more complex systems and exploring different numerical techniques to improve accuracy and computational efficiency.

References

[1] Richard Feynman. *Quantum Mechanics and Path Integrals*. McGraw-Hill, 1965.

[2] Jun John Sakurai. *Modern Quantum Mechanics*. Addison-Wesley, 2011.

[3] David J Griffiths. *Introduction to Quantum Mechanics*. Pearson Education, 2005.

[4] Lev Landau and Evgeny Lifshitz. *Quantum Mechanics: Non-Relativistic Theory*. Pergamon Press, 1977.

[5] N. Wheeler. Gaussian Wavepackets. https://www.reed.edu/physics/faculty/wheeler/documents/, 1998. [Accessed: 10-Jan-2023].

[6] Randall J Leveque. *Finite Difference Methods for Ordinary and Partial Differential Equations*. SIAM, 2007.

Code:

- Ahmet Burak Yıldırım: TDSE
- Yiğit Yaman: TISE

Appendix: MATLAB® Code for TISE

```
 %% Jacobi_1D
clear; close all; clc

psi_all = [];
for n = 1:9

    m = 1;          % mass of the particle (in amu)
    hbar = 1;       % reduced Planck's constant (in eV*fs)

    error = 1;
    i = 1;
    imax = 20000;
    Tol = 1e-6;



    % setup the grid

    dx = 0.1;       % spatial step size in the x direction (in nm)
    Lx = 10;        % size of the grid in the x direction (in nm)


    x = -Lx/2:dx:Lx/2;

    V = potential1D("harmonic", x) ;
    x = x';
    Nx = Lx/dx;
    psi_new = zeros(Nx+1,1);      % for Jacobi Method
    psi  = ones(Nx+1,1);
    dxs = dx^2;

    figure(2)
    hold on
    axis([0 imax 1e-6 10])

    set(gca,'YScale','log','fontsize',16,'fontname','Arial',...
        'XTick',0:imax/10:imax,'YTick',[1e-10 1e-8 1e-6 1e-4 1e-2 1 10])
    xlabel('Iteration','fontname','Arial','fontsize',20, 'fontweight', 'bold');
    ylabel('Absolute Error','fontname','Arial','fontsize',20, 'fontweight', 'bold');
    grid on
    box on


    while error > Tol && i < imax

        for j = 2:Nx
            psi_new(j) = hbar^2 * (psi(j-1) + psi(j+1))/(2*hbar^2+2*V(j)*m*dxs);
        end

        error = max (abs (psi_new-psi) );
        psi = psi_new;

        %         if mod(i,500)==0
        %             fprintf('iteration:  %30i\n' , i )
        %             fprintf('Absolute Error:  %1.10f\n' , error )
        %
        %             figure(2)
        %             plot(i,error, 'bo', 'linewidth',2)
        %             pause(0.1)
        %         end
```

```
        i = i+1;
    end

    psi = Normalizewave function1D(psi);
    psi_all = [psi_all psi];
end


%% Jacobi_1D plots
subplot(3,3,1)
plot(x,psi_all(:,1),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Jacobi (\omega = 0.01 eV)')

subplot(3,3,2)
plot(x,psi_all(:,2),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Jacobi (\omega = 0.05 eV)')

subplot(3,3,3)
plot(x,psi_all(:,3),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Jacobi (\omega = 0.25 eV)')

subplot(3,3,4)
plot(x,psi_all(:,4),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Jacobi (\omega = 1 eV)')

subplot(3,3,5)
plot(x,psi_all(:,5),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Jacobi (\omega = 5 eV)')

subplot(3,3,6)
plot(x,psi_all(:,6),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Jacobi (\omega = 25 eV)')

subplot(3,3,7)
plot(x,psi_all(:,7),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Jacobi (\omega = 100 eV)')

subplot(3,3,8)
plot(x,psi_all(:,8),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
```

```
title('1D Jacobi (\omega = 500 eV)')

subplot(3,3,9)
plot(x,psi_all(:,9),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Jacobi (\omega = 2500 eV)')


%% Gauss-Seidel_1D
clear; close all; clc

psi_all = [];
for n = 1:9
    m = 1;          % mass of the particle (in amu)
    hbar = 1;       % reduced Planck's constant (in eV*fs)

    error = 1;
    i = 1;
    imax = 20000;
    Tol = 1e-6;

    % setup the grid

    dx = 0.1;       % spatial step size in the x direction (in nm)
    Lx = 10;        % size of the grid in the x direction (in nm)
    x = -Lx/2:dx:Lx/2;
    V = potential1D("harmonic", x);
    x = x';
    Nx = Lx/dx;
    psi  = ones(Nx+1,1);
    psi(1) = 0;
    psi(end) = 0;
    dxs = dx^2;

    figure(2)
    hold on
    axis([0 imax 1e-6 10])

    set(gca,'YScale','log','fontsize',16,'fontname','Arial',...
        'XTick',0:imax/10:imax,'YTick',[1e-10 1e-8 1e-6 1e-4 1e-2 1 10])
    xlabel('Iteration','fontname','Arial','fontsize',20, 'fontweight', 'bold');
    ylabel('Absolute Error','fontname','Arial','fontsize',20, 'fontweight', 'bold');
    grid on
    box on


    while error > Tol && i < imax
        psi_old = psi;
        for j = 2:Nx
            psi(j) = hbar^2 * (psi(j-1) + psi(j+1))/(2*hbar^2+2*V(j)*m*dxs);
        end

        error = max (abs (psi-psi_old) );

        if mod(i,500)==0
            fprintf('iteration:  %30i\n' , i )
            fprintf('Absolute Error:  %1.10f\n' , error )
            figure(2)
            plot(i,error, 'bo', 'linewidth',2)
            pause(0.1)
        end
```

```
        i = i+1;
    end
    psi = Normalizewave function1D(psi);
    psi_all = [psi_all psi];
end

%% Gauss-Seidel_1D plots
subplot(3,3,1)
figure(1)
plot(x,psi_all(:,1),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Gauss-Seidel (\omega = 0.01 eV)')

subplot(3,3,2)
figure(1)
plot(x,psi_all(:,2),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Gauss-Seidel (\omega = 0.05 eV)')

subplot(3,3,3)
figure(1)
plot(x,psi_all(:,3),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Gauss-Seidel (\omega = 0.25 eV)')

subplot(3,3,4)
figure(1)
plot(x,psi_all(:,4),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Gauss-Seidel (\omega = 1 eV)')

subplot(3,3,5)
figure(1)
plot(x,psi_all(:,5),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Gauss-Seidel (\omega = 5 eV)')

subplot(3,3,6)
figure(1)
plot(x,psi_all(:,6),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Gauss-Seidel (\omega = 25 eV)')

subplot(3,3,7)
figure(1)
plot(x,psi_all(:,7),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Gauss-Seidel (\omega = 100 eV)')
```

```
subplot(3,3,8)
figure(1)
plot(x,psi_all(:,8),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Gauss-Seidel (\omega = 500 eV)')

subplot(3,3,9)
figure(1)
plot(x,psi_all(:,9),'LineWidth',2)
set(gca,'fontsize',14)
xlabel('x (nm)')
ylabel('|\psi|')
title('1D Gauss-Seidel (\omega = 2500 eV)')

%% Jacobi_2D
clear; close all; clc


psi_all = [];
for n = 1:9
    m = 1;          % mass of the particle (in amu)
    hbar = 1;       % reduced Planck's constant (in eV*fs)

    error = 1;
    i = 1;
    imax = 20000;
    Tol = 1e-6;

    % setup the grid

    dx = 0.1;       % spatial step size in the x direction (in nm)
    Lx = 10;        % size of the grid in the x direction (in nm)
    x = -Lx/2:dx:Lx/2;
    dy = 0.1;       % spatial step size in the y direction (in nm)
    Ly = 10;        % size of the grid in the y direction (in nm)
    y = -Ly/2:dy:Ly/2;

    x = x';
    y = y';
    V = potential2D("harmonic", x, y);


    Nx = Lx/dx;
    Ny = Ly/dy;
    psi_new = zeros(Nx+1,Ny+1);      % for Jacobi Method
    psi  = ones(Nx+1,Ny+1);
    dxs = dx^2;
    dys = dy^2;

    figure(2)
    hold on
    axis([0 imax 1e-6 10])

    set(gca,'YScale','log','fontsize',16,'fontname','Arial',...
        'XTick',0:imax/10:imax,'YTick',[1e-10 1e-8 1e-6 1e-4 1e-2 1 10])
    xlabel('Iteration','fontname','Arial','fontsize',20, 'fontweight', 'bold');
    ylabel('Absolute Error','fontname','Arial','fontsize',20, 'fontweight', 'bold');
    grid on
    box on
```

```
    while error > Tol && i < imax

        for j = 2:Nx
            for k = 2:Ny
                psi_new(j,k) = hbar^2 * (dys*(psi(j-1,k) + psi(j+1,k))+dxs*(psi(j,k-1) + psi(j,k+1)))...
                    /(2*hbar^2*dys+2*hbar^2*dxs+2*V(j,k)*m*dxs*dys);
            end
        end

        error = max(max (abs (psi_new-psi) ));
        psi = psi_new;

        %     if mod(i,500)==0
        %         fprintf('iteration:  %30i\n' , i )
        %         fprintf('Absolute Error:  %1.10f\n' , error )
        %         figure(2)
        %         plot(i,error, 'bo', 'linewidth',2)
        %         pause(0.1)
        %     end
        i = i+1;
    end

    % [X,Y] = meshgrid(x,y);

    psi = Normalizewave function2D(psi);
    psi_all = [psi_all psi];
end

% figure(1)
% for n = 1:size(x,1)
%     plot3(X(n,:),Y(n,:),psi(n,:),'b')
%     xlabel('x (nm)')
%     ylabel('y (nm)')
%     zlabel('|\psi|')
%     set(gca,'fontsize',14)
%     hold on
% end

% figure(3)
% imagesc(x,y,psi)
% xlabel('x (nm)')
% ylabel('y (nm)')
% title('2D Jacobi (\omega = 0.01 eV)')
% set(gca,'fontsize',14)
% colorbar

%% Jacobi_2D plots
subplot(3,3,1)
imagesc(x,y,psi_all(:,1:101))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Jacobi (\omega = 0.01 eV)')
set(gca,'fontsize',14)
colorbar

subplot(3,3,2)
imagesc(x,y,psi_all(:,102:202))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Jacobi (\omega = 0.05 eV)')
set(gca,'fontsize',14)
colorbar
```

```
subplot(3,3,3)
imagesc(x,y,psi_all(:,203:303))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Jacobi (\omega = 0.25 eV)')
set(gca,'fontsize',14)
colorbar

subplot(3,3,4)
imagesc(x,y,psi_all(:,304:404))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Jacobi (\omega = 1 eV)')
set(gca,'fontsize',14)
colorbar

subplot(3,3,5)
imagesc(x,y,psi_all(:,405:505))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Jacobi (\omega = 5 eV)')
set(gca,'fontsize',14)
colorbar

subplot(3,3,6)
imagesc(x,y,psi_all(:,506:606))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Jacobi (\omega = 25 eV)')
set(gca,'fontsize',14)
colorbar

subplot(3,3,7)
imagesc(x,y,psi_all(:,607:707))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Jacobi (\omega = 100 eV)')
set(gca,'fontsize',14)
colorbar

subplot(3,3,8)
imagesc(x,y,psi_all(:,708:808))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Jacobi (\omega = 500 eV)')
set(gca,'fontsize',14)
colorbar

subplot(3,3,9)
imagesc(x,y,psi_all(:,808:909))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Jacobi (\omega = 2500 eV)')
set(gca,'fontsize',14)
colorbar
%% Gauss-Seidel_2D
clear; close all; clc

psi_all = [];
for n = 1:9
    m = 1;          % mass of the particle (in amu)
    hbar = 1;       % reduced Planck's constant (in eV*fs)
```

```
error = 1;
i = 1;
imax = 20000;
Tol = 1e-6;

% setup the grid

dx = 0.1;      % spatial step size in the x direction (in nm)
Lx = 10;       % size of the grid in the x direction (in nm)
x = -Lx/2:dx:Lx/2;
dy = 0.1;      % spatial step size in the y direction (in nm)
Ly = 10;       % size of the grid in the y direction (in nm)
y = -Ly/2:dy:Ly/2;

x = x';
y = y';
V = potential2D("harmonic", x, y);


Nx = Lx/dx;
Ny = Ly/dy;
psi  = ones(Nx+1,Ny+1);
psi(1,1:end) = 0;
psi(1:end,1) = 0;
psi(end,1:end) = 0;
psi(1:end,end) = 0;
dxs = dx^2;
dys = dy^2;

figure(2)
hold on
axis([0 imax 1e-6 10])

set(gca,'YScale','log','fontsize',16,'fontname','Arial',...
    'XTick',0:imax/10:imax,'YTick',[1e-10 1e-8 1e-6 1e-4 1e-2 1 10])
xlabel('Iteration','fontname','Arial','fontsize',20, 'fontweight', 'bold');
ylabel('Absolute Error','fontname','Arial','fontsize',20, 'fontweight', 'bold');
grid on
box on


while error > Tol && i < imax
    psi_old = psi;
    for j = 2:Nx
        for k = 2:Ny
            psi(j,k) = hbar^2 * (dys*(psi(j-1,k) + psi(j+1,k))+dxs*(psi(j,k-1) + psi(j,k+1)))...
                /(2*hbar^2*dys+2*hbar^2*dxs+2*V(j,k)*m*dxs*dys);
        end
    end

    error = max(max (abs (psi-psi_old) ));


    %     if mod(i,500)==0
    %         fprintf('iteration:  %30i\n' , i )
    %         fprintf('Absolute Error:  %1.10f\n' , error )
    %         figure(2)
    %         plot(i,error, 'bo', 'linewidth',2)
    %         pause(0.1)
    %     end
    i = i+1;
end
```

```
    [X,Y] = meshgrid(x,y);

    psi = Normalizewave function2D(psi);
    psi_all = [psi_all psi];

end
% figure(1)

% for n = 1:size(x,1)
%     plot3(X(n,:),Y(n,:),psi(n,:),'b')
%     xlabel('x (nm)')
%     ylabel('y (nm)')
%     zlabel('|\psi|')
%     hold on
% end
% figure(3)
% imagesc(x,y,psi)
% xlabel('x (nm)')
% ylabel('y (nm)')
% title('2D Gauss-Seidel (\omega = 0.1 eV)')
% colorbar


%% Gauss-Seidel_2D plots
subplot(3,3,1)
imagesc(x,y,psi_all(:,1:101))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Gauss-Seidel (\omega = 0.01 eV)')
set(gca,'fontsize',14)
colorbar

subplot(3,3,2)
imagesc(x,y,psi_all(:,102:202))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Gauss-Seidel (\omega = 0.05 eV)')
set(gca,'fontsize',14)
colorbar

subplot(3,3,3)
imagesc(x,y,psi_all(:,203:303))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Gauss-Seidel (\omega = 0.25 eV)')
set(gca,'fontsize',14)
colorbar

subplot(3,3,4)
imagesc(x,y,psi_all(:,304:404))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Gauss-Seidel (\omega = 1 eV)')
set(gca,'fontsize',14)
colorbar

subplot(3,3,5)
imagesc(x,y,psi_all(:,405:505))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Gauss-Seidel (\omega = 5 eV)')
set(gca,'fontsize',14)
colorbar
```

```
subplot(3,3,6)
imagesc(x,y,psi_all(:,506:606))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Gauss-Seidel (\omega = 25 eV)')
set(gca,'fontsize',14)
colorbar

subplot(3,3,7)
imagesc(x,y,psi_all(:,607:707))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Gauss-Seidel (\omega = 100 eV)')
set(gca,'fontsize',14)
colorbar

subplot(3,3,8)
imagesc(x,y,psi_all(:,708:808))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Gauss-Seidel (\omega = 500 eV)')
set(gca,'fontsize',14)
colorbar

subplot(3,3,9)
imagesc(x,y,psi_all(:,808:909))
xlabel('x (nm)')
ylabel('y (nm)')
title('2D Gauss-Seidel (\omega = 2500 eV)')
set(gca,'fontsize',14)
colorbar


%% Functions
function V = potential1D(type, x)
if type == "harmonic"
    omega = input('Enter frequency'); % frequency of the harmonic oscillator (in eV)
end
switch type
    case "harmonic"
        % Define the potential energy function
        m = 1;
        X = reshape(x,[size(x,2) 1]); % 1D grid
        V = m*omega^2*(X.^2)/2; % harmonic oscillator potential
    case "finite-well"
        a = 8;
        V = zeros(size(x));
        V(abs(x) <= a/2) = 0;
        V(abs(x) > a/2) = 1; %inf;
    case "tunneling"
        a = 2.5;
        width = 0.1;
        V = zeros(size(x));
        V(x >= a) = 1;
        V(x > a+width) = 0;
    case "free-particle"
        V = zeros(size(x));
end
end
```

```
function psi = Normalizewave function1D(psi)
A = sqrt(trapz(psi.^2));
psi = psi/A;
end

function psi = Normalizewave function2D(psi)
A = sqrt(trapz(trapz(psi.^2)));
psi = psi/A;
end


function V = potential2D(type, x, y)

if type == "harmonic"
    omega = input('Enter frequency'); % frequency of the harmonic oscillator (in eV)
end
switch type
    case "harmonic"
        % Define the potential energy function
        %             omega = 0.7;
        m = 1;
        [X,Y] = meshgrid(x,y); % 2D grid
        V = m*omega^2*(X.^2+Y.^2)/2; % harmonic oscillator potential
    case "finite-well"
        a = 8;
        V = zeros(length(x),length(y));
        V(abs(x) <= a/2, abs(y) <= a/2) = 0;
        V(abs(x) > a/2, abs(y) > a/2) = 1; %inf;
    case "tunneling"
        a = 1;
        width_x = 0.5;
        V = zeros(length(x), length(y));
        V(x >= a, y >= a) = 5;
        V(x > a+width_x, y > a+width_x) = 0;
    case "single-slit"
        a = 1;
        dy = y(2) - y(1);
        width_x = 0.5;
        width_y = 1;
        V = zeros(length(x), length(y));
        V(x >= a, :) = 100;
        V(x >= a, end/2-width_y/dy/2:end/2+width_y/dy/2) = 0;
        V(x > a+width_x, :) = 0;
        V = transpose(V);
    case "double-slit"
        a = 1;
        dy = y(2) - y(1);
        width_x = 0.5;
        width_y = 1;
        V = zeros(length(x), length(y));
        V(x >= a, :) = 100;
        V(x >= a, end/2-width_y/dy:end/2-width_y/dy/2) = 0;
        V(x >= a, end/2+width_y/dy/2:end/2+width_y/dy) = 0;
        V(x > a+width_x, :) = 0;
        V = transpose(V);
    case "free-particle"
        V = zeros(length(x), length(y));
end
end
```

Appendix: MATLAB® Code for TDSE

```
%% Numerical analysis of Schrödinger equation
% i hbar d(psi(r,t))/dt = -(hbar^2)/(2m) nabla^2 psi(r,t) + V(r) psi(r,t)

% 1 dimensional -> r = (x):
% i hbar d(psi(x,t))/dt = -(hbar^2)/(2m) (d^2/dx^2) psi(x,t) + V(x) psi(x,t)

% 2 dimensional -> r = (x,y):
% i hbar d(psi(x,y,t))/dt = -(hbar^2)/(2m) (d^2/dx^2 + d^2/dy^2) psi(x,y,t) + V(x,y) psi(x,y,t)

%% Time-dependent Schrödinger equation
% Parabolic PDE

% OBSERVATIONS:
% EE: dt = 1e-4, tmax = 10, oscillations for finite-well,
% 1 - Explicit Euler
% 2 - Crank-Nicolson
%   2a - Jacobi
%   2b - Gauss-Seidel
%   2c - SOR
%   2d - ADI

clear; close all; clc;

% Initialization of parameters

dimension = 2; % 1 or 2

% Potential choices are applicable according to selected dimension
% List of available potentials:
% FOR 1D: harmonic / finite-well /  tunneling  / free-particle
% FOR 2D: harmonic / finite-well / double-slit / free-particle

potential1d = "tunneling"; % harmonic / finite-well / tunneling / free-particle
potential2d = "single-slit";   %double-slit / harmonic / finite-well / tunneling / free-particle / single-slit

m    = 1;         % mass of the particle (in amu)
hbar = 1;         % reduced Planck's constant (in eV*fs)

% Parameter table
% 1D - ExplicitEuler       : dt: TRUE: 2e-5 / CASES: 8e-5, 2e-4, 8e-4, 2e-3
% 1D - Crank-Nicolson      : dt: TRUE: 2e-5 / CASES: 8e-5, 2e-4, 8e-4, 2e-3
% 2D - ExplicitEuler       : dt: TRUE: 2e-5 / CASES: 8e-5, 2e-4, 8e-4, 2e-3
% 2D - Crank-Nicolson (ADI): dt: TRUE: 2e-5 / CASES: 8e-5, 2e-4, 8e-4, 2e-3

dt      = 8e-5;          % time step size (in fs)
dt_true = 2e-5;          % time step size for "true" solution (in fs)

tmax = 1;                % maximum time (in fs)

t = 0:dt:tmax;           % time grid
t_true = 0:dt_true:tmax; % time grid

if dimension == 1
    dx = 0.1;      % spatial step size in the x direction (in nm)
    Lx = 10;       % size of the grid in the x direction (in nm)
    x = -Lx/2:dx:Lx/2;
    X = reshape(x,[size(x,2) 1]); % 1D grid

    kx = 0.2;         % wave vector: for momentum
    r0 = -2; sigma = 0.5;
```

```
    psi_0 = exp(-0.5*(((X-r0))/sigma).^2).*exp(1i*(kx*(X)));
    psi_0 = Normalizewave function1D(psi_0);

    psi = zeros(length(x),length(t));
    % set the initial wave function
    psi(:,1) = psi_0;

    V = potential1D(potential1d,x);

    % TRUE SOLUTION (using smaller time step)

    psi_true = zeros(length(x),length(t_true));
    % To compare true solution psi_true with psi, creating a new arraw
    % and filling same time steps
    psi_true(:,1) = psi_0;

    % Solve the time-dependent Schrödinger equation using the finite difference method
    disp("Solving TDSE using the finite difference method")
    tic
    for k = 2:length(t)

        % d(psi(r,t))/dt = (i hbar)/(2m) nabla^2 psi(r,t) + 1/(i hbar) V(r) psi(r,t)
        % v = (i hbar)/(2m) -> diffusion coefficient
        % f = psi(r,t)
        % df/dt = v (d^2f/dx^2) + V(x)/(i hbar) f

        v = (1i*hbar)/(2*m); % diffusion coefficient
        psi_t = psi(:,k-1);

        pot = V/(1i*hbar);
        % Available solvers: ExplicitEuler1D / CrankNicolson1D
        psi(:,k) = CrankNicolson1D(psi_t, hbar, m, dx, dt, pot); % replace function
        psi(:,k) = Normalizewave function1D(psi(:,k));

    end
    toc
    disp("Solving for the true solution")
    tic
    % Solve for the true solution
    for k = 2:length(t_true)

        % d(psi(r,t))/dt = (i hbar)/(2m) nabla^2 psi(r,t) + 1/(i hbar) V(r) psi(r,t)
        % v = (i hbar)/(2m) -> diffusion coefficient
        % f = psi(r,t)
        % df/dt = v (d^2f/dx^2) + V(x)/(i hbar) f

        v = (1i*hbar)/(2*m); % diffusion coefficient
        psi_t = psi_true(:,k-1);

        pot = V/(1i*hbar);
        psi_true(:,k) = ExplicitEuler1D(psi_t, hbar, m, dx, dt_true, pot);
        psi_true(:,k) = Normalizewave function1D(psi_true(:,k));

    end
    toc
    psi_true_c = psi_true(:,1:length(t_true)/length(t):end);
end

if dimension == 2
    dx = 0.1;      % spatial step size in the x direction (in nm)
    dy = 0.1;      % spatial step size in the y direction (in nm)
    Lx = 10;       % size of the grid in the x direction (in nm)
    Ly = 10;       % size of the grid in the y direction (in nm)
```

```
    x = -Lx/2:dx:Lx/2;
    y = -Ly/2:dy:Ly/2;
    [X,Y] = meshgrid(x,y); % 2D grid

    k = [10, 0];   % wave vector: for momentum
    r0 = [-3, 0]; sigma = 0.5;
    psi_0 = exp(-((X-r0(1)).^2+(Y-r0(2)).^2)/(2*sigma)).*exp(1i*(k(1)*X+k(2)*Y));
    psi_0 = Normalizewave function2D(psi_0);
    psi = zeros(length(x),length(y),length(t));
    % set the initial wave function
    psi(:,:,1) = psi_0;

    V = potential2D(potential2d,x,y);

    % TRUE SOLUTION (using smaller time step)

    psi_true = zeros(length(x),length(y),length(t_true));
    % To compare true solution psi_true with psi, creating a new arraw
    % and filling same time steps
    psi_true(:,:,1) = psi_0;

    % Solve the time-dependent Schrödinger equation using the finite difference method
    disp("Solving TDSE using the finite difference method")
    tic
    for k = 2:length(t)

        % d(psi(r,t))/dt = (i hbar)/(2m) nabla^2 psi(r,t) + 1/(i hbar) V(r) psi(r,t)
        % v = (i hbar)/(2m) -> diffusion coefficient
        % f = psi(r,t)
        % df/dt = v (d^2f/dx^2 + d^2f/dy^2) + V(x,y)/(i hbar) f

        v = (1i*hbar)/(2*m); % diffusion coefficient
        psi_t = psi(:,:,k-1);

        pot = V/(1i*hbar);
        % Available solvers: ExplicitEuler2D / CrankNicolson2D (ADI)
        psi(:,:,k) = ExplicitEuler2D(psi_t, hbar, m, dx, dy, dt, pot);
        psi(:,:,k) = Normalizewave function2D(psi(:,:,k));

    end
    toc
    disp("Solving for the true solution")
    tic
    % Solve for the true solution
    for k = 2:length(t_true)

        % d(psi(r,t))/dt = (i hbar)/(2m) nabla^2 psi(r,t) + 1/(i hbar) V(r) psi(r,t)
        % v = (i hbar)/(2m) -> diffusion coefficient
        % f = psi(r,t)
        % df/dt = v (d^2f/dx^2 + d^2f/dy^2) + V(x,y)/(i hbar) f

        v = (1i*hbar)/(2*m); % diffusion coefficient
        psi_t = psi_true(:,:,k-1);

        pot = V/(1i*hbar);
        psi_true(:,:,k) = ExplicitEuler2D(psi_t, hbar, m, dx, dy, dt_true, pot);
        psi_true(:,:,k) = Normalizewave function2D(psi_true(:,:,k));

    end
    toc
    psi_true_c = psi_true(:,:,1:length(t_true)/length(t):end);
end
```

```
disp("Stability condition: alpha = " + dt*abs(v)/dx^2)

%% Save data: Dimension = 1, Method = Explicit Euler
% Run after each simulation

if dt == 8e-5
    X_1DEULER_dt8e5 = X;
    V_1DEULER_dt8e5 = V;
    t_1DEULER_dt8e5 = t;
    psi_1DEULER_dt8e5 = psi;
    psi_true_c_1DEULER_dt8e5 = psi_true_c;
    save 1DEULER_dt8e5.mat V_1DEULER_dt8e5 X_1DEULER_dt8e5 t_1DEULER_dt8e5 ...
        psi_1DEULER_dt8e5 psi_true_c_1DEULER_dt8e5
    clear
elseif dt == 2e-4
    X_1DEULER_dt2e4 = X;
    V_1DEULER_dt2e4 = V;
    t_1DEULER_dt2e4 = t;
    psi_1DEULER_dt2e4 = psi;
    psi_true_c_1DEULER_dt2e4 = psi_true_c;
    save 1DEULER_dt2e4.mat V_1DEULER_dt2e4 X_1DEULER_dt2e4 t_1DEULER_dt2e4 ...
        psi_1DEULER_dt2e4 psi_true_c_1DEULER_dt2e4
    clear
elseif dt == 8e-4
    X_1DEULER_dt8e4 = X;
    V_1DEULER_dt8e4 = V;
    t_1DEULER_dt8e4 = t;
    psi_1DEULER_dt8e4 = psi;
    psi_true_c_1DEULER_dt8e4 = psi_true_c;
    save 1DEULER_dt8e4.mat V_1DEULER_dt8e4 X_1DEULER_dt8e4 t_1DEULER_dt8e4 ...
        psi_1DEULER_dt8e4 psi_true_c_1DEULER_dt8e4
    clear
elseif dt == 2e-3
    X_1DEULER_dt2e3 = X;
    V_1DEULER_dt2e3 = V;
    t_1DEULER_dt2e3 = t;
    psi_1DEULER_dt2e3 = psi;
    psi_true_c_1DEULER_dt2e3 = psi_true_c;
    save 1DEULER_dt2e3.mat V_1DEULER_dt2e3 X_1DEULER_dt2e3 t_1DEULER_dt2e3 ...
        psi_1DEULER_dt2e3 psi_true_c_1DEULER_dt2e3
    clear
end

%% Save data: Dimension = 1, Method = Crank-Nicolson
% Run after each simulation

if dt == 8e-5
    X_1DCN_dt8e5 = X;
    V_1DCN_dt8e5 = V;
    t_1DCN_dt8e5 = t;
    psi_1DCN_dt8e5 = psi;
    psi_true_c_1DCN_dt8e5 = psi_true_c;
    save 1DCN_dt8e5.mat V_1DCN_dt8e5 X_1DCN_dt8e5 t_1DCN_dt8e5 psi_1DCN_dt8e5 psi_true_c_1DCN_dt8e5
    clear
elseif dt == 2e-4
    X_1DCN_dt2e4 = X;
    V_1DCN_dt2e4 = V;
    t_1DCN_dt2e4 = t;
    psi_1DCN_dt2e4 = psi;
    psi_true_c_1DCN_dt2e4 = psi_true_c;
    save 1DCN_dt2e4.mat V_1DCN_dt2e4 X_1DCN_dt2e4 t_1DCN_dt2e4 psi_1DCN_dt2e4 psi_true_c_1DCN_dt2e4
    clear
elseif dt == 8e-4
```

```
    X_1DCN_dt8e4 = X;
    V_1DCN_dt8e4 = V;
    t_1DCN_dt8e4 = t;
    psi_1DCN_dt8e4 = psi;
    psi_true_c_1DCN_dt8e4 = psi_true_c;
    save 1DCN_dt8e4.mat V_1DCN_dt8e4 X_1DCN_dt8e4 t_1DCN_dt8e4 psi_1DCN_dt8e4 psi_true_c_1DCN_dt8e4
    clear
elseif dt == 2e-3
    X_1DCN_dt2e3 = X;
    V_1DCN_dt2e3 = V;
    t_1DCN_dt2e3 = t;
    psi_1DCN_dt2e3 = psi;
    psi_true_c_1DCN_dt2e3 = psi_true_c;
    save 1DCN_dt2e3.mat V_1DCN_dt2e3 X_1DCN_dt2e3 t_1DCN_dt2e3 psi_1DCN_dt2e3 psi_true_c_1DCN_dt2e3
    clear
end

%% Figures for the Report: 1D TDSE: Euler method: ERROR

load('1DEULER_dt8e5.mat')
load('1DEULER_dt2e4.mat')
load('1DEULER_dt8e4.mat')
load('1DEULER_dt2e3.mat')

error_1DEULER_dt8e5 = abs(trapz(psi_1DEULER_dt8e5-psi_true_c_1DEULER_dt8e5,1));
error_1DEULER_dt2e4 = abs(trapz(psi_1DEULER_dt2e4-psi_true_c_1DEULER_dt2e4,1));
error_1DEULER_dt8e4 = abs(trapz(psi_1DEULER_dt8e4-psi_true_c_1DEULER_dt8e4,1));
error_1DEULER_dt2e3 = abs(trapz(psi_1DEULER_dt2e3-psi_true_c_1DEULER_dt2e3,1));

set(0, 'DefaultLineLineWidth', 2); close all;
tl = tiledlayout(1,2,'TileSpacing','tight','Padding','tight');
t1 = nexttile(1); hold on; grid on

plot(t_1DEULER_dt8e5,error_1DEULER_dt8e5)
plot(t_1DEULER_dt2e4,error_1DEULER_dt2e4)
plot(t_1DEULER_dt8e4,error_1DEULER_dt8e4)
plot(t_1DEULER_dt2e3,error_1DEULER_dt2e3)

legend("$\Delta t = 8\times10^{-5}$","$\Delta t = 2\times10^{-4}$",...
    "$\Delta t = 8\times10^{-4}$","$\Delta t = 2\times10^{-3}$")

t2 = nexttile(2); hold on; grid on

plot(t_1DEULER_dt8e5,error_1DEULER_dt8e5)
plot(t_1DEULER_dt2e4,error_1DEULER_dt2e4)
plot(t_1DEULER_dt8e4,error_1DEULER_dt8e4)
plot(t_1DEULER_dt2e3,error_1DEULER_dt2e3)

xlim([0 0.4])
ylim([0 0.2*1e-2])

title(t1,"(a) Error for explicit Euler method in 1D")
title(t2,"(b) Close-up to initial behavior of error")
title(tl,"POTENTIAL: Free-particle", 'Interpreter', 'latex')

xlabel(tl,'$\textrm{Time step}$','Interpreter','latex');
ylabel(tl,'$\textrm{Error}= |\int \psi - \psi_{true} \; dx|$','Interpreter','latex');

picturewidth = 3*34.4/3; hw_ratio = 0.2; fs = 13;
set(findall(gcf,'-property', 'FontSize'), 'FontSize', fs) % never change fontsize anymore!
set(findall(gcf,'-property', 'Box'), 'Box', 'on') % optional
set(findall(gcf, '-property', 'Interpreter'), 'Interpreter', 'latex')
set(findall(gcf, '-property', 'TickLabelInterpreter'), 'TickLabelInterpreter', 'latex')
```

```
set(gcf, 'Units', 'centimeters', 'Position', [2 1 picturewidth hw_ratio*picturewidth])
pos = get(gcf, 'Position');
set(findobj(gcf, 'Type', 'legend'), 'FontSize', fs-3, 'Location', 'northeast')
set(gca,'LooseInset',get(gca,'TightInset'))
set(gcf, 'PaperPositionMode', 'Auto', 'PaperUnits', 'centimeters', 'PaperSize', [pos(3), pos(4)])

%% Figures for the Report: 1D TDSE: Crank-Nicolson method: ERROR

load('1DCN_dt8e5.mat')
load('1DCN_dt2e4.mat')
load('1DCN_dt8e4.mat')
load('1DCN_dt2e3.mat')

error_1DCN_dt8e5 = abs(trapz(psi_1DCN_dt8e5-psi_true_c_1DCN_dt8e5,1));
error_1DCN_dt2e4 = abs(trapz(psi_1DCN_dt2e4-psi_true_c_1DCN_dt2e4,1));
error_1DCN_dt8e4 = abs(trapz(psi_1DCN_dt8e4-psi_true_c_1DCN_dt8e4,1));
error_1DCN_dt2e3 = abs(trapz(psi_1DCN_dt2e3-psi_true_c_1DCN_dt2e3,1));

set(0, 'DefaultLineLineWidth', 2); close all;
tl = tiledlayout(1,2,'TileSpacing','tight','Padding','tight');
t1 = nexttile(1); hold on; grid on

plot(t_1DCN_dt8e5,error_1DCN_dt8e5)
plot(t_1DCN_dt2e4,error_1DCN_dt2e4)
plot(t_1DCN_dt8e4,error_1DCN_dt8e4)
plot(t_1DCN_dt2e3,error_1DCN_dt2e3)

legend("$\Delta t = 8\times10^{-5}$","$\Delta t = 2\times10^{-4}$",...
    "$\Delta t = 8\times10^{-4}$","$\Delta t = 2\times10^{-3}$")

t2 = nexttile(2); hold on; grid on

plot(t_1DCN_dt8e5,error_1DCN_dt8e5)
plot(t_1DCN_dt2e4,error_1DCN_dt2e4)
plot(t_1DCN_dt8e4,error_1DCN_dt8e4)
plot(t_1DCN_dt2e3,error_1DCN_dt2e3)

xlim([0 0.4])
ylim([0 0.2*1e-2])

title(t1,"(a) Error for explicit Crank-Nicolson method in 1D")
title(t2,"(b) Close-up to initial behavior of error")
title(tl,"POTENTIAL: Free-particle", 'Interpreter', 'latex')

xlabel(tl,'$\textrm{Time step}$','Interpreter','latex');
ylabel(tl,'$\textrm{Error}= |\int \psi - \psi_{true} \; dx|$','Interpreter','latex');

picturewidth = 3*34.4/3; hw_ratio = 0.2; fs = 13;
set(findall(gcf,'-property', 'FontSize'), 'FontSize', fs) % never change fontsize anymore!
set(findall(gcf,'-property', 'Box'), 'Box', 'on') % optional
set(findall(gcf, '-property', 'Interpreter'), 'Interpreter', 'latex')
set(findall(gcf, '-property', 'TickLabelInterpreter'), 'TickLabelInterpreter', 'latex')
set(gcf, 'Units', 'centimeters', 'Position', [2 1 picturewidth hw_ratio*picturewidth])
pos = get(gcf, 'Position');
set(findobj(gcf, 'Type', 'legend'), 'FontSize', fs-3, 'Location', 'northwest')
set(gca,'LooseInset',get(gca,'TightInset'))
set(gcf, 'PaperPositionMode', 'Auto', 'PaperUnits', 'centimeters', 'PaperSize', [pos(3), pos(4)])

%% Figures for the Report: 1D TDSE: Euler method: SNAPSHOTS

load('1DEULER_dt8e5.mat')
load('1DEULER_dt2e4.mat')
load('1DEULER_dt8e4.mat')
```

```
load('1DEULER_dt2e3.mat')

set(0, 'DefaultLineLineWidth', 2); lw = 2;

close all; f = figure;
tl = tiledlayout(3,1,'TileSpacing','tight','Padding','tight');

X = X_1DEULER_dt8e5; V = V_1DEULER_dt8e5;
V(1,1) = 1; V(1,end) = 1; % for visualization purposes

t1 = nexttile(1); hold on; grid on; set(gcf,'color','w');
potplot1 = plot(X, V,'-r','LineWidth',lw);
psiplot1 = plot(X,abs(psi_true_c_1DEULER_dt8e5(:,1)),'-k','LineWidth',lw);
realplot1 = plot(X,real(psi_true_c_1DEULER_dt8e5(:,1)),'-b','LineWidth',lw*0.5);
imagplot1 = plot(X,imag(psi_true_c_1DEULER_dt8e5(:,1)),'-g','LineWidth',lw*0.5);
ylim([-0.5 0.5])

t2 = nexttile(2); hold on; grid on; set(gcf,'color','w');
potplot2 = plot(X, V,'-r','LineWidth',lw);
psiplot2 = plot(X,abs(psi_1DEULER_dt2e4(:,1)),'-k','LineWidth',lw);
realplot2 = plot(X,real(psi_1DEULER_dt2e4(:,1)),'-b','LineWidth',lw*0.5);
imagplot2 = plot(X,imag(psi_1DEULER_dt2e4(:,1)),'-g','LineWidth',lw*0.5);
ylim([-0.5 0.5])

t3 = nexttile(3); hold on; grid on; set(gcf,'color','w');
potplot3 = plot(X, V,'-r','LineWidth',lw);
psiplot3 = plot(X,abs(psi_1DEULER_dt8e4(:,1)),'-k','LineWidth',lw);
realplot3 = plot(X,real(psi_1DEULER_dt8e4(:,1)),'-b','LineWidth',lw*0.5);
imagplot3 = plot(X,imag(psi_1DEULER_dt8e4(:,1)),'-g','LineWidth',lw*0.5);

ylim([-0.5 0.5])
legend("V(x)","|$\psi$|","Re($\psi$)","Im($\psi$)")
picturewidth = 3*34.4/3/4; hw_ratio = 0.2*3*4; fs = 13;

no_snap = 5;
for snap = 1:no_snap

    delete(potplot1)
    delete(psiplot1)
    delete(realplot1)
    delete(imagplot1)
    delete(potplot2)
    delete(psiplot2)
    delete(realplot2)
    delete(imagplot2)
    delete(potplot3)
    delete(psiplot3)
    delete(realplot3)
    delete(imagplot3)

    ts_1 = (length(psi_true_c_1DEULER_dt8e5)-1) / (no_snap-1);
    ts_2 = (length(psi_1DEULER_dt2e4)-1) / (no_snap-1);
    ts_3 = (length(psi_1DEULER_dt8e4)-1) / (no_snap-1);

    t1 = nexttile(1); hold on; grid on; set(gcf,'color','w');
    potplot1 = plot(X, V,'-r','LineWidth',lw);
    psiplot1 = plot(X,abs(psi_true_c_1DEULER_dt8e5(:,(snap-1)*ts_1+1)),'-k','LineWidth',lw);
    realplot1 = plot(X,real(psi_true_c_1DEULER_dt8e5(:,(snap-1)*ts_1+1)),'-b','LineWidth',lw*0.5);
    imagplot1 = plot(X,imag(psi_true_c_1DEULER_dt8e5(:,(snap-1)*ts_1+1)),'-g','LineWidth',lw*0.5);

    t2 = nexttile(2); hold on; grid on; set(gcf,'color','w');
    potplot2 = plot(X, V,'-r','LineWidth',lw);
    psiplot2 = plot(X,abs(psi_1DEULER_dt2e4(:,(snap-1)*ts_2+1)),'-k','LineWidth',lw);
```

```
    realplot2 = plot(X,real(psi_1DEULER_dt2e4(:,(snap-1)*ts_2+1)),'-b','LineWidth',lw*0.5);
    imagplot2 = plot(X,imag(psi_1DEULER_dt2e4(:,(snap-1)*ts_2+1)),'-g','LineWidth',lw*0.5);

    t3 = nexttile(3); hold on; grid on; set(gcf,'color','w');
    potplot3 = plot(X, V,'-r','LineWidth',lw);
    psiplot3 = plot(X,abs(psi_1DEULER_dt8e4(:,(snap-1)*ts_3+1)),'-k','LineWidth',lw);
    realplot3 = plot(X,real(psi_1DEULER_dt8e4(:,(snap-1)*ts_3+1)),'-b','LineWidth',lw*0.5);
    imagplot3 = plot(X,imag(psi_1DEULER_dt8e4(:,(snap-1)*ts_3+1)),'-g','LineWidth',lw*0.5);

    xlabel('x (nm)');
    tx = ['Time step = ', num2str(4/(no_snap-1)*snap-1)];

    legend("V(x)","$|\psi|$","Re($\psi$)","Im($\psi$)",'NumColumns',2)
    title(t1, tx, 'Interpreter', 'latex')
    set(findall(gcf,'-property', 'FontSize'), 'FontSize', fs) % never change fontsize anymore!
    set(findall(gcf,'-property', 'Box'), 'Box', 'on') % optional
    set(findall(gcf, '-property', 'Interpreter'), 'Interpreter', 'latex')
    set(findall(gcf, '-property', 'TickLabelInterpreter'), 'TickLabelInterpreter', 'latex')
    set(gcf, 'Units', 'centimeters', 'Position', [2 1 picturewidth hw_ratio*picturewidth])
    pos = get(gcf, 'Position');
    set(findobj(gcf, 'Type', 'legend'), 'FontSize', fs-3, 'Location', 'southoutside')
    set(gca,'LooseInset',get(gca,'TightInset'))
    set(gcf, 'PaperPositionMode', 'Auto', 'PaperUnits', 'centimeters', 'PaperSize', [pos(3), pos(4)])
    ftx = "1DEULER_" + snap;
    drawnow
    print(f,ftx,'-dpdf','-vector', '-fillpage')

end

%% Figures for the Report: 1D TDSE: Crank-Nicolson method: SNAPSHOTS

load('1DCN_dt8e5.mat')
load('1DCN_dt2e4.mat')
load('1DCN_dt8e4.mat')
load('1DCN_dt2e3.mat')

set(0, 'DefaultLineLineWidth', 2); lw = 2;

close all; f = figure;
tl = tiledlayout(3,1,'TileSpacing','tight','Padding','tight');

X = X_1DCN_dt8e5; V = V_1DCN_dt8e5;
V(1,1) = 1; V(1,end) = 1; % for visualization purposes

t1 = nexttile(1); hold on; grid on; set(gcf,'color','w');
potplot1 = plot(X, V,'-r','LineWidth',lw);
psiplot1 = plot(X,abs(psi_true_c_1DCN_dt8e5(:,1)),'-k','LineWidth',lw);
realplot1 = plot(X,real(psi_true_c_1DCN_dt8e5(:,1)),'-b','LineWidth',lw*0.5);
imagplot1 = plot(X,imag(psi_true_c_1DCN_dt8e5(:,1)),'-g','LineWidth',lw*0.5);
ylim([-0.5 0.5])

t2 = nexttile(2); hold on; grid on; set(gcf,'color','w');
potplot2 = plot(X, V,'-r','LineWidth',lw);
psiplot2 = plot(X,abs(psi_1DCN_dt2e4(:,1)),'-k','LineWidth',lw);
realplot2 = plot(X,real(psi_1DCN_dt2e4(:,1)),'-b','LineWidth',lw*0.5);
imagplot2 = plot(X,imag(psi_1DCN_dt2e4(:,1)),'-g','LineWidth',lw*0.5);
ylim([-0.5 0.5])

t3 = nexttile(3); hold on; grid on; set(gcf,'color','w');
potplot3 = plot(X, V,'-r','LineWidth',lw);
psiplot3 = plot(X,abs(psi_1DCN_dt8e4(:,1)),'-k','LineWidth',lw);
realplot3 = plot(X,real(psi_1DCN_dt8e4(:,1)),'-b','LineWidth',lw*0.5);
imagplot3 = plot(X,imag(psi_1DCN_dt8e4(:,1)),'-g','LineWidth',lw*0.5);
```

```
ylim([-0.5 0.5])
legend("V(x)","|$\psi$|","Re($\psi$)","Im($\psi$)")
picturewidth = 3*34.4/3/4; hw_ratio = 0.2*3*4; fs = 13;

no_snap = 5;
for snap = 1:no_snap

    delete(potplot1)
    delete(psiplot1)
    delete(realplot1)
    delete(imagplot1)
    delete(potplot2)
    delete(psiplot2)
    delete(realplot2)
    delete(imagplot2)
    delete(potplot3)
    delete(psiplot3)
    delete(realplot3)
    delete(imagplot3)

    ts_1 = (length(psi_true_c_1DCN_dt8e5)-1) / (no_snap-1);
    ts_2 = (length(psi_1DCN_dt2e4)-1) / (no_snap-1);
    ts_3 = (length(psi_1DCN_dt8e4)-1) / (no_snap-1);

    t1 = nexttile(1); hold on; grid on; set(gcf,'color','w');
    potplot1 = plot(X, V,'-r','LineWidth',lw);
    psiplot1 = plot(X,abs(psi_true_c_1DCN_dt8e5(:,(snap-1)*ts_1+1)),'-k','LineWidth',lw);
    realplot1 = plot(X,real(psi_true_c_1DCN_dt8e5(:,(snap-1)*ts_1+1)),'-b','LineWidth',lw*0.5);
    imagplot1 = plot(X,imag(psi_true_c_1DCN_dt8e5(:,(snap-1)*ts_1+1)),'-g','LineWidth',lw*0.5);

    t2 = nexttile(2); hold on; grid on; set(gcf,'color','w');
    potplot2 = plot(X, V,'-r','LineWidth',lw);
    psiplot2 = plot(X,abs(psi_1DCN_dt2e4(:,(snap-1)*ts_2+1)),'-k','LineWidth',lw);
    realplot2 = plot(X,real(psi_1DCN_dt2e4(:,(snap-1)*ts_2+1)),'-b','LineWidth',lw*0.5);
    imagplot2 = plot(X,imag(psi_1DCN_dt2e4(:,(snap-1)*ts_2+1)),'-g','LineWidth',lw*0.5);

    t3 = nexttile(3); hold on; grid on; set(gcf,'color','w');
    potplot3 = plot(X, V,'-r','LineWidth',lw);
    psiplot3 = plot(X,abs(psi_1DCN_dt8e4(:,(snap-1)*ts_3+1)),'-k','LineWidth',lw);
    realplot3 = plot(X,real(psi_1DCN_dt8e4(:,(snap-1)*ts_3+1)),'-b','LineWidth',lw*0.5);
    imagplot3 = plot(X,imag(psi_1DCN_dt8e4(:,(snap-1)*ts_3+1)),'-g','LineWidth',lw*0.5);

    xlabel('x (nm)');
    tx = ['Time step = ', num2str(4/(no_snap-1)*snap-1)];

    legend("V(x)","$|\psi|$","Re($\psi$)","Im($\psi$)",'NumColumns',2)
    title(t1, tx, 'Interpreter', 'latex')
    set(findall(gcf,'-property', 'FontSize'), 'FontSize', fs) % never change fontsize anymore!
    set(findall(gcf,'-property', 'Box'), 'Box', 'on') % optional
    set(findall(gcf, '-property', 'Interpreter'), 'Interpreter', 'latex')
    set(findall(gcf, '-property', 'TickLabelInterpreter'), 'TickLabelInterpreter', 'latex')
    set(gcf, 'Units', 'centimeters', 'Position', [2 1 picturewidth hw_ratio*picturewidth])
    pos = get(gcf, 'Position');
    set(findobj(gcf, 'Type', 'legend'), 'FontSize', fs-3, 'Location', 'southoutside')
    set(gca,'LooseInset',get(gca,'TightInset'))
    set(gcf, 'PaperPositionMode', 'Auto', 'PaperUnits', 'centimeters', 'PaperSize', [pos(3), pos(4)])
    ftx = "1DCN_" + snap;
    drawnow
    print(f,ftx,'-dpdf','-vector', '-fillpage')

end
```

```
%% Figures for the Report: 1D TDSE: Tunneling: SNAPSHOTS

load('1DEULER_dt8e4.mat')
load('1DCN_dt8e4.mat')

set(0, 'DefaultLineLineWidth', 2); lw = 2;

close all; f = figure;
tl = tiledlayout(3,1,'TileSpacing','tight','Padding','tight');

X = X_1DEULER_dt8e4; V = V_1DEULER_dt8e4;
V(1,1) = 1; V(1,end) = 1; % for visualization purposes

t1 = nexttile(1); hold on; grid on; set(gcf,'color','w');
potplot1 = plot(X, V,'-r','LineWidth',lw);
psiplot1 = plot(X,abs(psi_true_c_1DEULER_dt8e4(:,1)),'-k','LineWidth',lw);
realplot1 = plot(X,real(psi_true_c_1DEULER_dt8e4(:,1)),'-b','LineWidth',lw*0.5);
imagplot1 = plot(X,imag(psi_true_c_1DEULER_dt8e4(:,1)),'-g','LineWidth',lw*0.5);
ylim([-0.5 0.5])

t2 = nexttile(2); hold on; grid on; set(gcf,'color','w');
potplot2 = plot(X, V,'-r','LineWidth',lw);
psiplot2 = plot(X,abs(psi_1DEULER_dt8e4(:,1)),'-k','LineWidth',lw);
realplot2 = plot(X,real(psi_1DEULER_dt8e4(:,1)),'-b','LineWidth',lw*0.5);
imagplot2 = plot(X,imag(psi_1DEULER_dt8e4(:,1)),'-g','LineWidth',lw*0.5);
ylim([-0.5 0.5])

t3 = nexttile(3); hold on; grid on; set(gcf,'color','w');
potplot3 = plot(X, V,'-r','LineWidth',lw);
psiplot3 = plot(X,abs(psi_1DCN_dt8e4(:,1)),'-k','LineWidth',lw);
realplot3 = plot(X,real(psi_1DCN_dt8e4(:,1)),'-b','LineWidth',lw*0.5);
imagplot3 = plot(X,imag(psi_1DCN_dt8e4(:,1)),'-g','LineWidth',lw*0.5);

ylim([-0.5 0.5])
legend("V(x)","|$\psi$|","Re($\psi$)","Im($\psi$)")
picturewidth = 3*34.4/3/4; hw_ratio = 0.2*3*4; fs = 13;

no_snap = 5;
for snap = 1:no_snap

    delete(potplot1)
    delete(psiplot1)
    delete(realplot1)
    delete(imagplot1)
    delete(potplot2)
    delete(psiplot2)
    delete(realplot2)
    delete(imagplot2)
    delete(potplot3)
    delete(psiplot3)
    delete(realplot3)
    delete(imagplot3)

    ts_1 = (length(psi_true_c_1DEULER_dt8e4)-1) / (no_snap-1);
    ts_2 = (length(psi_1DEULER_dt8e4)-1) / (no_snap-1);
    ts_3 = (length(psi_1DCN_dt8e4)-1) / (no_snap-1);

    t1 = nexttile(1); hold on; grid on; set(gcf,'color','w');
    potplot1 = plot(X, V,'-r','LineWidth',lw);
    psiplot1 = plot(X,abs(psi_true_c_1DEULER_dt8e4(:,(snap-1)*ts_1+1)),'-k','LineWidth',lw);
    realplot1 = plot(X,real(psi_true_c_1DEULER_dt8e4(:,(snap-1)*ts_1+1)),'-b','LineWidth',lw*0.5);
    imagplot1 = plot(X,imag(psi_true_c_1DEULER_dt8e4(:,(snap-1)*ts_1+1)),'-g','LineWidth',lw*0.5);
```

```
    t2 = nexttile(2); hold on; grid on; set(gcf,'color','w');
    potplot2 = plot(X, V,'-r','LineWidth',lw);
    psiplot2 = plot(X,abs(psi_1DEULER_dt8e4(:,(snap-1)*ts_2+1)),'-k','LineWidth',lw);
    realplot2 = plot(X,real(psi_1DEULER_dt8e4(:,(snap-1)*ts_2+1)),'-b','LineWidth',lw*0.5);
    imagplot2 = plot(X,imag(psi_1DEULER_dt8e4(:,(snap-1)*ts_2+1)),'-g','LineWidth',lw*0.5);

    t3 = nexttile(3); hold on; grid on; set(gcf,'color','w');
    potplot3 = plot(X, V,'-r','LineWidth',lw);
    psiplot3 = plot(X,abs(psi_1DCN_dt8e4(:,(snap-1)*ts_3+1)),'-k','LineWidth',lw);
    realplot3 = plot(X,real(psi_1DCN_dt8e4(:,(snap-1)*ts_3+1)),'-b','LineWidth',lw*0.5);
    imagplot3 = plot(X,imag(psi_1DCN_dt8e4(:,(snap-1)*ts_3+1)),'-g','LineWidth',lw*0.5);

    xlabel('x (nm)');
    tx = ['Time step = ', num2str(4/(no_snap-1)*snap-1)];

    legend("V(x)","$|\psi|$","Re($\psi$)","Im($\psi$)",'NumColumns',2)
    title(t1, tx, 'Interpreter', 'latex')
    set(findall(gcf,'-property', 'FontSize'), 'FontSize', fs) % never change fontsize anymore!
    set(findall(gcf,'-property', 'Box'), 'Box', 'on') % optional
    set(findall(gcf, '-property', 'Interpreter'), 'Interpreter', 'latex')
    set(findall(gcf, '-property', 'TickLabelInterpreter'), 'TickLabelInterpreter', 'latex')
    set(gcf, 'Units', 'centimeters', 'Position', [2 1 picturewidth hw_ratio*picturewidth])
    pos = get(gcf, 'Position');
    set(findobj(gcf, 'Type', 'legend'), 'FontSize', fs-3, 'Location', 'southoutside')
    set(gca,'LooseInset',get(gca,'TightInset'))
    set(gcf, 'PaperPositionMode', 'Auto', 'PaperUnits', 'centimeters', 'PaperSize', [pos(3), pos(4)])
    ftx = "1Dcomparison_" + snap;
    drawnow
    print(f,ftx,'-dpdf','-vector', '-fillpage')

end

%% Save data: Dimension = 2, Method = Explicit Euler
% Run after each simulation

if dt == 8e-5
    X_2DEULER_dt8e5 = X;
    Y_2DEULER_dt8e5 = Y;
    V_2DEULER_dt8e5 = V;
    t_2DEULER_dt8e5 = t;
    psi_2DEULER_dt8e5 = psi;
    psi_true_c_2DEULER_dt8e5 = psi_true_c;
    save 2DEULER_dt8e5.mat V_2DEULER_dt8e5 X_2DEULER_dt8e5 Y_2DEULER_dt8e5 t_2DEULER_dt8e5 ...
        psi_2DEULER_dt8e5 psi_true_c_2DEULER_dt8e5
    clear
elseif dt == 2e-4
    X_2DEULER_dt2e4 = X;
    Y_2DEULER_dt2e4 = Y;
    V_2DEULER_dt2e4 = V;
    t_2DEULER_dt2e4 = t;
    psi_2DEULER_dt2e4 = psi;
    psi_true_c_2DEULER_dt2e4 = psi_true_c;
    save 2DEULER_dt2e4.mat V_2DEULER_dt2e4 X_2DEULER_dt2e4 Y_2DEULER_dt2e4 t_2DEULER_dt2e4 ...
        psi_2DEULER_dt2e4 psi_true_c_2DEULER_dt2e4
    clear
elseif dt == 8e-4
    X_2DEULER_dt8e4 = X;
    Y_2DEULER_dt8e4 = Y;
    V_2DEULER_dt8e4 = V;
    t_2DEULER_dt8e4 = t;
    psi_2DEULER_dt8e4 = psi;
    psi_true_c_2DEULER_dt8e4 = psi_true_c;
    save 2DEULER_dt8e4.mat V_2DEULER_dt8e4 X_2DEULER_dt8e4 Y_2DEULER_dt8e4 t_2DEULER_dt8e4 ...
```

```
        psi_2DEULER_dt8e4 psi_true_c_2DEULER_dt8e4
    clear
elseif dt == 2e-3
    X_2DEULER_dt2e3 = X;
    Y_2DEULER_dt2e3 = Y;
    V_2DEULER_dt2e3 = V;
    t_2DEULER_dt2e3 = t;
    psi_2DEULER_dt2e3 = psi;
    psi_true_c_2DEULER_dt2e3 = psi_true_c;
    save 2DEULER_dt2e3.mat V_2DEULER_dt2e3 X_2DEULER_dt2e3 Y_2DEULER_dt2e3 t_2DEULER_dt2e3 ...
        psi_2DEULER_dt2e3 psi_true_c_2DEULER_dt2e3
    clear
end

%% Save data: Dimension = 2, Method = Crank-Nicolson
% Run after each simulation

if dt == 8e-5
    X_2DCN_dt8e5 = X;
    Y_2DCN_dt8e5 = Y;
    V_2DCN_dt8e5 = V;
    t_2DCN_dt8e5 = t;
    psi_2DCN_dt8e5 = psi;
    psi_true_c_2DCN_dt8e5 = psi_true_c;
    save 2DCN_dt8e5.mat V_2DCN_dt8e5 X_2DCN_dt8e5 Y_2DCN_dt8e5 t_2DCN_dt8e5 psi_2DCN_dt8e5 psi_true_c_2DCN_dt8e5
    clear
elseif dt == 2e-4
    X_2DCN_dt2e4 = X;
    Y_2DCN_dt2e4 = Y;
    V_2DCN_dt2e4 = V;
    t_2DCN_dt2e4 = t;
    psi_2DCN_dt2e4 = psi;
    psi_true_c_2DCN_dt2e4 = psi_true_c;
    save 2DCN_dt2e4.mat V_2DCN_dt2e4 X_2DCN_dt2e4 Y_2DCN_dt2e4 t_2DCN_dt2e4 psi_2DCN_dt2e4 psi_true_c_2DCN_dt2e4
    clear
elseif dt == 8e-4
    X_2DCN_dt8e4 = X;
    Y_2DCN_dt8e4 = Y;
    V_2DCN_dt8e4 = V;
    t_2DCN_dt8e4 = t;
    psi_2DCN_dt8e4 = psi;
    psi_true_c_2DCN_dt8e4 = psi_true_c;
    save 2DCN_dt8e4.mat V_2DCN_dt8e4 X_2DCN_dt8e4 Y_2DCN_dt8e4 t_2DCN_dt8e4 psi_2DCN_dt8e4 psi_true_c_2DCN_dt8e4
    clear
elseif dt == 2e-3
    X_2DCN_dt2e3 = X;
    Y_2DCN_dt2e3 = Y;
    V_2DCN_dt2e3 = V;
    t_2DCN_dt2e3 = t;
    psi_2DCN_dt2e3 = psi;
    psi_true_c_2DCN_dt2e3 = psi_true_c;
    save 2DCN_dt2e3.mat V_2DCN_dt2e3 X_2DCN_dt2e3 Y_2DCN_dt2e3 t_2DCN_dt2e3 psi_2DCN_dt2e3 psi_true_c_2DCN_dt2e3
    clear
end

%% Figures for the Report: 2D TDSE: Euler method: ERROR

load('2DEULER_dt8e5.mat')
load('2DEULER_dt2e4.mat')
load('2DEULER_dt8e4.mat')
load('2DEULER_dt2e3.mat')

error_2DEULER_dt8e5 = abs(trapz(trapz(psi_2DEULER_dt8e5-psi_true_c_2DEULER_dt8e5,1)));
```

```
error_2DEULER_dt2e4 = abs(trapz(trapz(psi_2DEULER_dt2e4-psi_true_c_2DEULER_dt2e4,1)));
error_2DEULER_dt8e4 = abs(trapz(trapz(psi_2DEULER_dt8e4-psi_true_c_2DEULER_dt8e4,1)));
error_2DEULER_dt2e3 = abs(trapz(trapz(psi_2DEULER_dt2e3-psi_true_c_2DEULER_dt2e3,1)));

error_2DEULER_dt8e5 = reshape(error_2DEULER_dt8e5,[length(error_2DEULER_dt8e5) 1]);
error_2DEULER_dt2e4 = reshape(error_2DEULER_dt2e4,[length(error_2DEULER_dt2e4) 1]);
error_2DEULER_dt8e4 = reshape(error_2DEULER_dt8e4,[length(error_2DEULER_dt8e4) 1]);
error_2DEULER_dt2e3 = reshape(error_2DEULER_dt2e3,[length(error_2DEULER_dt2e3) 1]);

set(0, 'DefaultLineLineWidth', 2); close all;
tl = tiledlayout(1,2,'TileSpacing','tight','Padding','tight');
t1 = nexttile(1); hold on; grid on

plot(t_2DEULER_dt8e5,error_2DEULER_dt8e5)
plot(t_2DEULER_dt2e4,error_2DEULER_dt2e4)
plot(t_2DEULER_dt8e4,error_2DEULER_dt8e4)
plot(t_2DEULER_dt2e3,error_2DEULER_dt2e3)

legend("$\Delta t = 8\times10^{-5}$","$\Delta t = 2\times10^{-4}$",...
    "$\Delta t = 8\times10^{-4}$","$\Delta t = 2\times10^{-3}$")

t2 = nexttile(2); hold on; grid on

plot(t_2DEULER_dt8e5,error_2DEULER_dt8e5)
plot(t_2DEULER_dt2e4,error_2DEULER_dt2e4)
plot(t_2DEULER_dt8e4,error_2DEULER_dt8e4)
plot(t_2DEULER_dt2e3,error_2DEULER_dt2e3)

xlim([0 0.1])
%ylim([0 0.2*1e-2])

title(t1,"(a) Error for explicit Euler method in 2D")
title(t2,"(b) Close-up to initial behavior of error")
title(tl,"POTENTIAL: Free-particle", 'Interpreter', 'latex')

xlabel(tl,'$\textrm{Time step}$','Interpreter','latex');
ylabel(tl,'$\textrm{Error}= |\int \int \psi - \psi_{true} \; dx dy|$','Interpreter','latex');

picturewidth = 3*34.4/3; hw_ratio = 0.2; fs = 13;
set(findall(gcf,'-property', 'FontSize'), 'FontSize', fs) % never change fontsize anymore!
set(findall(gcf,'-property', 'Box'), 'Box', 'on') % optional
set(findall(gcf, '-property', 'Interpreter'), 'Interpreter', 'latex')
set(findall(gcf, '-property', 'TickLabelInterpreter'), 'TickLabelInterpreter', 'latex')
set(gcf, 'Units', 'centimeters', 'Position', [2 1 picturewidth hw_ratio*picturewidth])
pos = get(gcf, 'Position');
set(findobj(gcf, 'Type', 'legend'), 'FontSize', fs-3, 'Location', 'northeast')
set(gca,'LooseInset',get(gca,'TightInset'))
set(gcf, 'PaperPositionMode', 'Auto', 'PaperUnits', 'centimeters', 'PaperSize', [pos(3), pos(4)])

%% Figures for the Report: 2D TDSE: Crank-Nicolson method: ERROR

load('2DCN_dt8e5.mat')
load('2DCN_dt2e4.mat')
load('2DCN_dt8e4.mat')
load('2DCN_dt2e3.mat')

error_2DCN_dt8e5 = abs(trapz(trapz(psi_2DCN_dt8e5-psi_true_c_2DCN_dt8e5,1)));
error_2DCN_dt2e4 = abs(trapz(trapz(psi_2DCN_dt2e4-psi_true_c_2DCN_dt2e4,1)));
error_2DCN_dt8e4 = abs(trapz(trapz(psi_2DCN_dt8e4-psi_true_c_2DCN_dt8e4,1)));
error_2DCN_dt2e3 = abs(trapz(trapz(psi_2DCN_dt2e3-psi_true_c_2DCN_dt2e3,1)));

error_2DCN_dt8e5 = reshape(error_2DCN_dt8e5,[length(error_2DCN_dt8e5) 1]);
error_2DCN_dt2e4 = reshape(error_2DCN_dt2e4,[length(error_2DCN_dt2e4) 1]);
```

```
error_2DCN_dt8e4 = reshape(error_2DCN_dt8e4,[length(error_2DCN_dt8e4) 1]);
error_2DCN_dt2e3 = reshape(error_2DCN_dt2e3,[length(error_2DCN_dt2e3) 1]);

set(0, 'DefaultLineLineWidth', 2); close all;
tl = tiledlayout(1,2,'TileSpacing','tight','Padding','tight');
t1 = nexttile(1); hold on; grid on

plot(t_2DCN_dt8e5,error_2DCN_dt8e5)
plot(t_2DCN_dt2e4,error_2DCN_dt2e4)
plot(t_2DCN_dt8e4,error_2DCN_dt8e4)
plot(t_2DCN_dt2e3,error_2DCN_dt2e3)

legend("$\Delta t = 8\times10^{-5}$","$\Delta t = 2\times10^{-4}$",...
    "$\Delta t = 8\times10^{-4}$","$\Delta t = 2\times10^{-3}$")

t2 = nexttile(2); hold on; grid on

plot(t_2DCN_dt8e5,error_2DCN_dt8e5)
plot(t_2DCN_dt2e4,error_2DCN_dt2e4)
plot(t_2DCN_dt8e4,error_2DCN_dt8e4)
plot(t_2DCN_dt2e3,error_2DCN_dt2e3)

xlim([0 0.1])
%ylim([0 0.2*1e-2])

title(t1,"(a) Error for Crank-Nicolson method in 2D")
title(t2,"(b) Close-up to initial behavior of error")
title(tl,"POTENTIAL: Free-particle", 'Interpreter', 'latex')

xlabel(tl,'$\textrm{Time step}$','Interpreter','latex');
ylabel(tl,'$\textrm{Error}= |\int \int \psi - \psi_{true} \; dx dy|$','Interpreter','latex');

picturewidth = 3*34.4/3; hw_ratio = 0.2; fs = 13;
set(findall(gcf,'-property', 'FontSize'), 'FontSize', fs) % never change fontsize anymore!
set(findall(gcf,'-property', 'Box'), 'Box', 'on') % optional
set(findall(gcf, '-property', 'Interpreter'), 'Interpreter', 'latex')
set(findall(gcf, '-property', 'TickLabelInterpreter'), 'TickLabelInterpreter', 'latex')
set(gcf, 'Units', 'centimeters', 'Position', [2 1 picturewidth hw_ratio*picturewidth])
pos = get(gcf, 'Position');
set(findobj(gcf, 'Type', 'legend'), 'FontSize', fs-3, 'Location', 'northwest')
set(gca,'LooseInset',get(gca,'TightInset'))
set(gcf, 'PaperPositionMode', 'Auto', 'PaperUnits', 'centimeters', 'PaperSize', [pos(3), pos(4)])

%% Figures for the Report: 2D TDSE: Euler method: SNAPSHOTS

load('2DEULER_dt8e5.mat')
load('2DEULER_dt2e4.mat')
load('2DEULER_dt8e4.mat')
load('2DEULER_dt2e3.mat')

x = X_2DEULER_dt2e3(1,:); y = Y_2DEULER_dt2e3(:,1);

set(0, 'DefaultLineLineWidth', 2); lw = 2; fs = 13;

close all; f = figure;
tl = tiledlayout(3,1,'TileSpacing','tight','Padding','tight');

X = X_2DEULER_dt8e5; Y = Y_2DEULER_dt8e5; V = V_2DEULER_dt8e5;
V(:,1) = 1; V(:,end) = 1; V(1,:) = 1; V(end,:) = 1; % for visualization purposes

t1 = nexttile(1); hold on; set(gcf,'color','w');
potplot1 = imagesc(x,y,V);
psiplot1 = imagesc(x,y,abs(psi_true_c_2DEULER_dt2e3(:,:,1)));
```

```
t2 = nexttile(2); hold on; set(gcf,'color','w');
potplot2 = imagesc(x,y,V);
psiplot2 = imagesc(x,y,abs(psi_2DEULER_dt2e4(:,:,1)));

t3 = nexttile(3); hold on; set(gcf,'color','w');
potplot3 = imagesc(x,y,V);
psiplot3 = imagesc(x,y,abs(psi_2DEULER_dt8e4(:,:,1)));
c = colorbar; clim([0 25]); c.Location = "southoutside";

picturewidth = 3*34.4/3/4; hw_ratio = 0.2*3*4;

no_snap = 6;
for snap = 1:no_snap

    delete(potplot1)
    delete(psiplot1)
    delete(potplot2)
    delete(psiplot2)
    delete(potplot3)
    delete(psiplot3)

    ts_1 = (length(psi_true_c_2DEULER_dt2e3)-1) / (no_snap-1);
    ts_2 = (length(psi_2DEULER_dt2e4)-1) / (no_snap-1);
    ts_3 = (length(psi_2DEULER_dt8e4)-1) / (no_snap-1);

    t1 = nexttile(1); hold on; set(gcf,'color','w');
potplot1 = imagesc(x,y,V);
psiplot1 = imagesc(x,y,abs(psi_true_c_2DEULER_dt2e3(:,:,(snap-1)*ts_1+1)));

    t2 = nexttile(2); hold on; set(gcf,'color','w');
potplot2 = imagesc(x,y,V);
psiplot2 = imagesc(x,y,abs(psi_2DEULER_dt2e4(:,:,(snap-1)*ts_2+1)));
    ylabel('y (nm)');

    t3 = nexttile(3); hold on; set(gcf,'color','w');
potplot3 = imagesc(x,y,V);
psiplot3 = imagesc(x,y,abs(psi_2DEULER_dt8e4(:,:,(snap-1)*ts_3+1)));
    %c = colorbar; clim([0 25]); c.Location = "southoutside";

    xlabel('x (nm)');
    tx = ['Time step = ', num2str(1/(no_snap-1)*(snap-1))];

    title(t1, tx, 'Interpreter', 'latex')
    set(findall(gcf,'-property', 'FontSize'), 'FontSize', fs) % never change fontsize anymore!
    set(findall(gcf,'-property', 'Box'), 'Box', 'on') % optional
    set(findall(gcf, '-property', 'Interpreter'), 'Interpreter', 'latex')
    set(findall(gcf, '-property', 'TickLabelInterpreter'), 'TickLabelInterpreter', 'latex')
    set(gcf, 'Units', 'centimeters', 'Position', [2 1 picturewidth hw_ratio*picturewidth])
    pos = get(gcf, 'Position');
    set(findobj(gcf, 'Type', 'legend'), 'FontSize', fs-3, 'Location', 'southoutside')
    set(gca,'LooseInset',get(gca,'TightInset'))
    set(gcf, 'PaperPositionMode', 'Auto', 'PaperUnits', 'centimeters', 'PaperSize', [pos(3), pos(4)])
    ftx = "2DEULER_" + snap;
    drawnow
    print(f,ftx,'-dpdf','-vector', '-fillpage')

end

%% Figures for the Report: 2D TDSE: Crank-Nicolson method: SNAPSHOTS

load('2DEULER_dt2e3.mat')
load('2DCN_dt8e5.mat')
```

```
load('2DCN_dt2e4.mat')
load('2DCN_dt8e4.mat')
load('2DCN_dt2e3.mat')

x = X_2DCN_dt2e3(1,:); y = Y_2DCN_dt2e3(:,1);

set(0, 'DefaultLineLineWidth', 2); lw = 2; fs = 13;

close all; f = figure;
tl = tiledlayout(3,1,'TileSpacing','tight','Padding','tight');

X = X_2DCN_dt8e5; Y = Y_2DCN_dt8e5; V = V_2DCN_dt8e5;
V(:,1) = 1; V(:,end) = 1; V(1,:) = 1; V(end,:) = 1; % for visualization purposes

t1 = nexttile(1); hold on; set(gcf,'color','w');
potplot1 = imagesc(x,y,V);
psiplot1 = imagesc(x,y,abs(psi_true_c_2DEULER_dt2e3(:,:,1)));

t2 = nexttile(2); hold on; set(gcf,'color','w');
potplot2 = imagesc(x,y,V);
psiplot2 = imagesc(x,y,abs(psi_2DCN_dt2e4(:,:,1)));

t3 = nexttile(3); hold on; set(gcf,'color','w');
potplot3 = imagesc(x,y,V);
psiplot3 = imagesc(x,y,abs(psi_2DCN_dt8e4(:,:,1)));
c = colorbar; clim([0 25]); c.Location = "southoutside";

picturewidth = 3*34.4/3/4; hw_ratio = 0.2*3*4;

no_snap = 6;
for snap = 1:no_snap

    delete(potplot1)
    delete(psiplot1)
    delete(potplot2)
    delete(psiplot2)
    delete(potplot3)
    delete(psiplot3)

    ts_1 = (length(psi_true_c_2DEULER_dt2e3)-1) / (no_snap-1);
    ts_2 = (length(psi_2DCN_dt2e4)-1) / (no_snap-1);
    ts_3 = (length(psi_2DCN_dt8e4)-1) / (no_snap-1);

    t1 = nexttile(1); hold on; set(gcf,'color','w');
potplot1 = imagesc(x,y,V);
psiplot1 = imagesc(x,y,abs(psi_true_c_2DEULER_dt2e3(:,:,(snap-1)*ts_1+1)));

    t2 = nexttile(2); hold on; set(gcf,'color','w');
potplot2 = imagesc(x,y,V);
psiplot2 = imagesc(x,y,abs(psi_2DCN_dt2e4(:,:,(snap-1)*ts_2+1)));
    ylabel('y (nm)');

    t3 = nexttile(3); hold on; set(gcf,'color','w');
potplot3 = imagesc(x,y,V);
psiplot3 = imagesc(x,y,abs(psi_2DCN_dt8e4(:,:,(snap-1)*ts_3+1)));
    %c = colorbar; clim([0 25]); c.Location = "southoutside";

    xlabel('x (nm)');
    tx = ['Time step = ', num2str(1/(no_snap-1)*(snap-1))];

    title(t1, tx, 'Interpreter', 'latex')
    set(findall(gcf,'-property', 'FontSize'), 'FontSize', fs) % never change fontsize anymore!
    set(findall(gcf,'-property', 'Box'), 'Box', 'on') % optional
```

```
    set(findall(gcf, '-property', 'Interpreter'), 'Interpreter', 'latex')
    set(findall(gcf, '-property', 'TickLabelInterpreter'), 'TickLabelInterpreter', 'latex')
    set(gcf, 'Units', 'centimeters', 'Position', [2 1 picturewidth hw_ratio*picturewidth])
    pos = get(gcf, 'Position');
    set(findobj(gcf, 'Type', 'legend'), 'FontSize', fs-3, 'Location', 'southoutside')
    set(gca,'LooseInset',get(gca,'TightInset'))
    set(gcf, 'PaperPositionMode', 'Auto', 'PaperUnits', 'centimeters', 'PaperSize', [pos(3), pos(4)])
    ftx = "2DCN_" + snap;
    drawnow
    print(f,ftx,'-dpdf','-vector', '-fillpage')

end

%% Figures for the Report: 2D TDSE: Double/single-slit: SNAPSHOTS

load('2DEULER_dt8e5_doubleslit.mat')

X_2DEULER_dt8e5_doubleslit = X_2DEULER_dt8e5;
Y_2DEULER_dt8e5_doubleslit = Y_2DEULER_dt8e5;
V_2DEULER_dt8e5_doubleslit = V_2DEULER_dt8e5;
psi_true_c_2DEULER_dt8e5_doubleslit = psi_true_c_2DEULER_dt8e5;
psi_2DEULER_dt8e5_doubleslit = psi_2DEULER_dt8e5;

clear X_2DEULER_dt8e5 Y_2DEULER_dt8e5 V_2DEULER_dt8e5 psi_true_c_2DEULER_dt8e5 psi_2DEULER_dt8e5

load('2DEULER_dt8e5_singleslit.mat')

X_2DEULER_dt8e5_singleslit = X_2DEULER_dt8e5;
Y_2DEULER_dt8e5_singleslit = Y_2DEULER_dt8e5;
V_2DEULER_dt8e5_singleslit = V_2DEULER_dt8e5;
psi_true_c_2DEULER_dt8e5_singleslit = psi_true_c_2DEULER_dt8e5;
psi_2DEULER_dt8e5_singleslit = psi_2DEULER_dt8e5;

clear X_2DEULER_dt8e5 Y_2DEULER_dt8e5 V_2DEULER_dt8e5 psi_true_c_2DEULER_dt8e5 psi_2DEULER_dt8e5
%%
x = X_2DEULER_dt8e5_doubleslit(1,:); y = Y_2DEULER_dt8e5_doubleslit(:,1);

set(0, 'DefaultLineLineWidth', 2); lw = 2; fs = 13;

close all; f = figure;
tl = tiledlayout(2,1,'TileSpacing','tight','Padding','tight');

%V_2DEULER_dt8e5_doubleslit(:,1) = 1; V_2DEULER_dt8e5_doubleslit(:,end) = 1;
%V_2DEULER_dt8e5_doubleslit(1,:) = 1; V_2DEULER_dt8e5_doubleslit(end,:) = 1; % for visualization purposes
%V_2DEULER_dt8e5_singleslit(:,1) = 1; V_2DEULER_dt8e5_singleslit(:,end) = 1;
%V_2DEULER_dt8e5_singleslit(1,:) = 1; V_2DEULER_dt8e5_singleslit(end,:) = 1; % for visualization purposes

t1 = nexttile(1); hold on; set(gcf,'color','w');
potplot1 = imagesc(x,y,V_2DEULER_dt8e5_singleslit,'AlphaData', 0.5);
psiplot1 = imagesc(x,y,abs(psi_true_c_2DEULER_dt8e5_singleslit(:,:,1)),'AlphaData', 0.5);
c = colorbar; c.Location = "southoutside";
%clim([0 25]);

t2 = nexttile(2); hold on; set(gcf,'color','w');
potplot2 = imagesc(x,y,V_2DEULER_dt8e5_doubleslit,'AlphaData', 0.5);
psiplot2 = imagesc(x,y,abs(psi_true_c_2DEULER_dt8e5_doubleslit(:,:,1)),'AlphaData', 0.5);

c = colorbar; c.Location = "southoutside";
% clim([0 25]);

picturewidth = 3*34.4/3/4; hw_ratio = 0.2*3*4;

no_snap = 6;
```

```
for snap = 1:no_snap

    delete(potplot1)
    delete(psiplot1)
    delete(potplot2)
    delete(psiplot2)

    ts_1 = (length(psi_2DEULER_dt8e5_singleslit)-1) / (no_snap-1);
    ts_2 = (length(psi_2DEULER_dt8e5_doubleslit)-1) / (no_snap-1);

    t1 = nexttile(1); hold on; set(gcf,'color','w');
potplot1 = imagesc(x,y,V_2DEULER_dt8e5_singleslit,'AlphaData', 0.5);
psiplot1 = imagesc(x,y,abs(psi_true_c_2DEULER_dt8e5_singleslit(:,:,(snap-1)*ts_1+1)),'AlphaData', 0.5);
    caxis([0 max(max(abs(psi_true_c_2DEULER_dt8e5_singleslit(:,:,(snap-1)*ts_1+1))))])

    t2 = nexttile(2); hold on; set(gcf,'color','w');
potplot2 = imagesc(x,y,V_2DEULER_dt8e5_doubleslit,'AlphaData', 0.5);
psiplot2 = imagesc(x,y,abs(psi_true_c_2DEULER_dt8e5_doubleslit(:,:,(snap-1)*ts_2+1)),'AlphaData', 0.5);
    caxis([0 max(max(abs(psi_true_c_2DEULER_dt8e5_doubleslit(:,:,(snap-1)*ts_1+1))))])

    ylabel(tl, 'y (nm)');

    xlabel('x (nm)');
    tx = ['Time step = ', num2str(1/(no_snap-1)*(snap-1))];

    title(t1, tx, 'Interpreter', 'latex')
    set(findall(gcf,'-property', 'FontSize'), 'FontSize', fs) % never change fontsize anymore!
    set(findall(gcf,'-property', 'Box'), 'Box', 'on') % optional
    set(findall(gcf, '-property', 'Interpreter'), 'Interpreter', 'latex')
    set(findall(gcf, '-property', 'TickLabelInterpreter'), 'TickLabelInterpreter', 'latex')
    set(gcf, 'Units', 'centimeters', 'Position', [2 1 picturewidth hw_ratio*picturewidth])
    pos = get(gcf, 'Position');
    set(findobj(gcf, 'Type', 'legend'), 'FontSize', fs-3, 'Location', 'southoutside')
    set(gca,'LooseInset',get(gca,'TightInset'))
    set(gcf, 'PaperPositionMode', 'Auto', 'PaperUnits', 'centimeters', 'PaperSize', [pos(3), pos(4)])
    ftx = "2DSLIT_" + snap;
    drawnow
    print(f,ftx,'-dpdf','-vector', '-fillpage')

end

%% Functions

function f_next = ExplicitEuler1D(f, hbar, m, dx, dt, V)

    % d(psi(r,t))/dt = (i hbar)/(2m) nabla^2 psi(r,t) + 1/(i hbar) V(r) psi(r,t)
    % v = (i hbar)/(2m) -> diffusion coefficient
    % f = psi(r,t)
    % df/dt = v (d^2f/dx^2) + V(x)/(i hbar) f

    f_next = 0*f;

    for x_ = 2:size(f,1)-1
        laplacian_x = 1/dx^2 * (f(x_-1) - 2*f(x_) + f(x_+1));
        f_next(x_) = f(x_) + (-hbar^2/(2*m))/(1i*hbar)*dt*(laplacian_x) + dt*V(x_)*f(x_)/(1i*hbar);
    end
end

function f_next = RungeKutta41D(f, hbar, m, dx, dt, V)

    % d(psi(r,t))/dt = (i hbar)/(2m) nabla^2 psi(r,t) + 1/(i hbar) V(r) psi(r,t)
    % v = (i hbar)/(2m) -> diffusion coefficient
    % f = psi(r,t)
```

```
    % df/dt = v (d^2f/dx^2) + V(x)/(i hbar) f

    f_next = 0*f;

    for x_ = 2:size(f,1)-1
        laplacian_x = 1/dx^2 * (f(x_-1) - 2*f(x_) + f(x_+1));
        f_next(x_) = f(x_) + (-hbar^2/(2*m))/(1i*hbar)*dt*(laplacian_x) + dt*V(x_)*f(x_)/(1i*hbar);
    end
end

function f_new = CrankNicolson1D(f, hbar, m, dx, dt, V)
    SoR = 1.8;

    N = length(f);
    f_temp = 0*f;
    f_new = f_temp;

    a1 = -(1i * hbar * dt)/(4*m*dx^2);
    b1 = (2*1i*hbar*dt)/(4*m*dx^2) + 1 - (V*dt)/(2*1i*hbar);
    c1 = a1;

    a2 = (1i * hbar * dt)/(4*m*dx^2);
    b2 = 1 - (2*1i*hbar*dt)/(4*m*dx^2) + (V*dt)/(2*1i*hbar);
    c2 = a2;

    A = zeros(N-2,N-2);
    B = zeros(N-2,N-2);

    for x_ = 1:N-2
        for y_ = 1:N-2
            if x_-y_ == -1
                A(x_,y_) = a1;
                B(x_,y_) = a2;
            elseif x_-y_ == 0
                A(x_,y_) = b1(x_);
                B(x_,y_) = b2(x_);
            elseif x_-y_ == +1
                A(x_,y_) = c1;
                B(x_,y_) = c2;
            end
        end
    end

    tol = 1e-6;
    max_iter = 1;

    for i = 1:max_iter
        RHS = B*f(2:N-1);
        f_new(2:N-1) = A\RHS;
        if norm(f_new - f, 2) < tol
            break
        end
    end
end

function f_new = CrankNicolson2D(f, hbar, m, dx, dy, dt, V)

    N = size(f,1);
    f_new = 0*f;

    a1_hs = -(1i * hbar * dt)/(4*m*dx^2);
    b1_hs = 1 + (2*1i*hbar*dt)/(4*m*dx^2) - (V*dt)/(4*1i*hbar);
    c1_hs = a1_hs;
```

```
    a2_hs = (1i * hbar * dt)/(4*m*dy^2);
    b2_hs = 1 - (2*1i*hbar*dt)/(4*m*dy^2) + (V*dt)/(4*1i*hbar);
    c2_hs = a2_hs;

    a1_fs = -(1i * hbar * dt)/(4*m*dy^2);
    b1_fs = 1 + (2*1i*hbar*dt)/(4*m*dy^2) - (V*dt)/(4*1i*hbar);
    c1_fs = a1_fs;
    a2_fs = (1i * hbar * dt)/(4*m*dx^2);
    b2_fs = 1 - (2*1i*hbar*dt)/(4*m*dx^2) + (V*dt)/(4*1i*hbar);
    c2_fs = a2_fs;

    A_hs = zeros(N-2,N-2);
    B_hs = zeros(N-2,N-2);
    A_fs = zeros(N-2,N-2);
    B_fs = zeros(N-2,N-2);

    for x_ = 1:N-2
        for y_ = 1:N-2
            if x_-y_ == -1
                A_hs(x_,y_) = A_hs(x_,y_) + a1_hs;
                B_hs(x_,y_) = B_hs(x_,y_) + a2_hs;

                A_fs(x_,y_) = A_fs(x_,y_) + c1_fs;
                B_fs(x_,y_) = B_fs(x_,y_) + c2_fs;
            elseif x_-y_ == 0
                A_hs(x_,y_) = A_hs(x_,y_) + b1_hs(x_,y_);
                B_hs(x_,y_) = B_hs(x_,y_) + b2_hs(x_,y_);

                A_fs(x_,y_) = A_fs(x_,y_) + b1_fs(x_,y_);
                B_fs(x_,y_) = B_fs(x_,y_) + b2_fs(x_,y_);
            elseif x_-y_ == +1
                A_hs(x_,y_) = A_hs(x_,y_) + c1_hs;
                B_hs(x_,y_) = B_hs(x_,y_) + c2_hs;

                A_fs(x_,y_) = A_fs(x_,y_) + a1_fs;
                B_fs(x_,y_) = B_fs(x_,y_) + a2_fs;
            end
        end
    end

    tol = 1e-6;
    max_iter = 1; %1000


    % Not yet iterative
    for i = 1:max_iter
        % First x-direction
        RHS_hs = B_hs*f(2:N-1,2:N-1)';
        % Then y-direction
        f_hs = (A_hs\RHS_hs);
        %f_hs = f(2:N-1,2:N-1);
        RHS_fs = B_fs*f_hs';
        f_new(2:N-1,2:N-1) = (A_fs\RHS_fs);
        f_new = f_new;
%         if norm(f_new - f, 2) < tol
%             break
%         end
        f = f_new;
    end

end

function f_next = ExplicitEuler2D(f, hbar, m, dx, dy, dt, V)
```

```
    f_next = 0*f;
    for x_ = 2:size(f,1)-1
        for y_ = 2:size(f,2)-1
            laplacian_x = 1/dx^2 * (f(x_-1,y_) - 2*f(x_,y_) + f(x_+1,y_));
            laplacian_y = 1/dy^2 * (f(x_,y_-1) - 2*f(x_,y_) + f(x_,y_+1));
            f_next(x_,y_) = f(x_,y_) + (-hbar^2/(2*m))/(1i*hbar)*dt*(laplacian_x + laplacian_y) + ...
                dt*V(x_,y_)*f(x_,y_)/(1i*hbar);
        end
    end
end

function psi = Normalizewave function1D(psi)
    A = sqrt(trapz(psi.^2));
    psi = psi/A;
end

function psi = Normalizewave function2D(psi)
    A = sqrt(trapz(trapz(psi.^2)));
    psi = psi/A;
end

function V = potential1D(type, x)
    switch type
        case "harmonic"
            % Define the potential energy function
            omega = 0.1; % frequency of the harmonic oscillator (in eV)
            m = 1;
            X = reshape(x,[size(x,2) 1]); % 1D grid
            V = m*omega^2*(X.^2)/2; % harmonic oscillator potential
        case "finite-well"
            a = 8;
            V = zeros(size(x));
            V(abs(x) <= a/2) = 0;
            V(abs(x) > a/2) = 1; %inf;
        case "tunneling"
            a = 1;
            width = 0.5;
            V = zeros(size(x));
            V(x >= a) = 0.5;
            V(x > a+width) = 0;
        case "free-particle"
            V = zeros(size(x));
    end
end

function V = potential2D(type, x, y)
    switch type
        case "harmonic"
            % Define the potential energy function
            omega = 0.1; % frequency of the harmonic oscillator (in eV)
            m = 1;
            [X,Y] = meshgrid(x,y); % 2D grid
            V = m*omega^2*(X.^2+Y.^2)/2; % harmonic oscillator potential
        case "finite-well"
            a = 8;
            V = zeros(length(x),length(y));
            V(abs(x) <= a/2, abs(y) <= a/2) = 0;
            V(abs(x) > a/2, abs(y) > a/2) = 1; %inf;
        case "tunneling"
            a = 1;
            width_x = 0.5;
            V = zeros(length(x), length(y));
            V(x >= a, y >= a) = 5;
```

```
            V(x > a+width_x, y > a+width_x) = 0;
        case "single-slit"
            a = 1;
            dy = y(2) - y(1);
            width_x = 0.5;
            width_y = 1;
            V = zeros(length(x), length(y));
            V(x >= a, :) = 100;
            V(x >= a, round((end+1)/2-width_y/dy/2):round((end+1)/2+width_y/dy/2)) = 0;
            V(x > a+width_x, :) = 0;
            V = transpose(V);
        case "double-slit"
            a = 1;
            %dx = x(2) - x(1);
            dy = y(2) - y(1);
            width_x = 0.5;
            width_y = 1;
            V = zeros(length(x), length(y));
            V(x >= a, :) = 100;
            V(x >= a, round((end+1)/2-width_y/dy):round((end+1)/2-width_y/dy/2)) = 0;
            V(x >= a, round((end+1)/2+width_y/dy/2):round((end+1)/2+width_y/dy)) = 0;
            V(x > a+width_x, :) = 0;
            V = transpose(V);
        case "free-particle"
            V = zeros(length(x), length(y));
    end
end
```