Effects of external potentials on the solutions of linear Schrödinger equation

Amina Darayeva
Supervisor: Amin Esfahani
Second reader: Alejandro Javier Castro Castilla
Department of Mathematics
Nazarbayev University

June 2, 2025

Abstract

The linear Schrödinger equation is a fundamental equation in quantum mechanics, which describes the dynamics of a particle's wave function in space and time. This project focuses on numerically studying how the addition of an external potential influences the motion of the wave function governed by the linear Schrödinger equation on the real line. We implement Python codes and compare several numerical methods - Fourier, Chebyshev, Crank-Nicolson, Leapfrog, Lax-Wendroff Methods as well as Duhamel's pinciple and Time-discretization method-to study their performance in terms of its accuracy, stability and time efficiency. By analyzing how solutions change under attractive ($\varepsilon > 0$) and repulsive ($\varepsilon < 0$) forces, we gain insight into the interaction between wave behavior and potential energy, as well as the effectiveness of different numerical approaches for such systems.

Contents

1 Introduction

The Schrödinger equation was published by physicist Erwin Schrödinger in 1926, which had a significant impact on the understanding of microscopic physics. In contrast with classical mechanics, which describes particles through definite trajectories, quantum mechanics models particles as wave functions by partial differential equations (Feit et al., 1982). The linear Schrödinger equation in one spatial dimension is written:

$$iu_t = u_{xx} + V(x)u$$

where $u(x,t)$ is the complex-valued wave function, and $V(x)$ represents an external potential acting on the system.

In this project, we focus on a simplified case:

$$iu_t = u_{xx} + \varepsilon x^2 u,$$

where $\varepsilon \in \mathbb{R}$. In this form, we are able to study attractive potentials (when $\varepsilon > 0$), as well as repulsive potentials (when $\varepsilon < 0$). The presence of an external potential significantly affects the behavior of the solution: it can lead to stable bound states or cause rapid spread depending on the sign and magnitude of ε.

In terms of mathematics, the Schrödinger equation is a diffusive and norm-preserving partial differential equation that conserves the L^2-norm of the wave function over time (Feit et al., 1982).

We evaluated the performance of several standard methods: Chebyshev Spectral Method, Fourier Split-Step Method, Crank-Nicolson Method, Lax–Wendroff and Leapfrog Methods, as well as Duhamel's Principle and Time-discretization.

The main goal of this project is to numerically study the effects of an additional external force on the solutions of the linear Schrödinger equation on the real line. By varying the parameter ε and applying different numerical approaches, we aim to draw conclusions about the behavior of the wave function under external influence, and to identify suitable methods for accurate and stable solution in quantum systems with potentials.

2 Exact Solution

We consider the Schrödinger equation:

$$iu_t = u_{xx} + \varepsilon x^2 u,$$

with the initial condition:

$$u(x,0) = \operatorname{sech}(x).$$

We use the separation of variables in the form of:

$$u(x,t) = g(t)A(x).$$

Substituting into the equation:

$$ig'(t)A(x) = g(t)A''(x) + \varepsilon x^2 g(t)A(x),$$

$$\frac{ig'(t)}{g(t)} = \frac{A''(x)}{A(x)} + \varepsilon x^2.$$

Since the left side depends only on t and the right only on x, both must equal a separation constant, for example $-\alpha$:

$$\frac{ig'}{g} = -\alpha \qquad ig' + \alpha g = 0,$$

$$g(t) = e^{i\alpha t}.$$

The spatial equation becomes:

$$A''(x) + (\varepsilon x^2 - \alpha)A(x) = 0.$$

Then, we introduce a new variable:

$$\xi = \varepsilon^{1/4} x \qquad x^2 = \xi^2/\sqrt{\varepsilon}, \quad \frac{d^2A}{dx^2} = \sqrt{\varepsilon}\frac{d^2A}{d\xi^2}.$$

This transforms the spatial equation into:

$$A_{\xi\xi} + (\xi^2 - \lambda)A = 0, \quad \text{where} \quad \lambda = \frac{\alpha}{\sqrt{\varepsilon}}.$$

This is the Hermite differential equation (second-order linear ordinary differential equation). The solutions are:

$$A_n(x) = H_n(\varepsilon^{1/4}x)e^{-\frac{\sqrt{\varepsilon}}{2}x^2}, \quad \lambda_n = 2n+1.$$

where H_n - Hermite polynomials, which form an orthogonal basis for the space of square-integrable functions on $\mathbb{R}$ (Dattoli et al., 1994)

Thus,

$$\alpha_n = (2n+1)\sqrt{\varepsilon}, \quad g_n(t) = e^{i\alpha_n t}.$$

So, the general solution is:

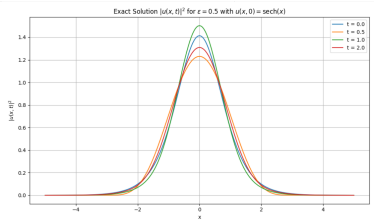
$$u(x,t) = \sum_{n=0}^{\infty} c_n H_n(\varepsilon^{1/4} x) e^{-\frac{\sqrt{\varepsilon}}{2}x^2} e^{i(2n+1)\sqrt{\varepsilon}t},$$

where the coefficients c_n are computed via projection onto the initial condition:

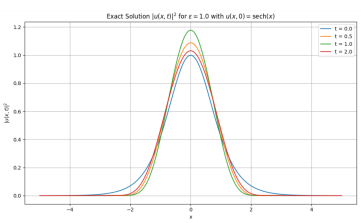
$$c_n = \int_{-\infty}^{\infty} \operatorname{sech}(x) \cdot H_n(x)\, dx.$$

2.1 Graphical Representation

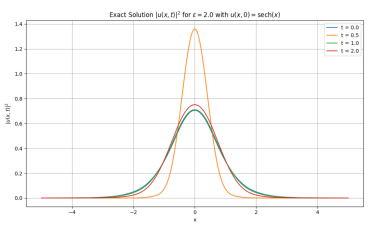
Below is a plot that shows the dynamics of $|u(x,t)|^2$ over time, starting from the initial condition $\operatorname{sech}(x)$, approximated by Hermite functions and computed using the first 20 terms. As time progresses, the wave function illustrates stable oscillatory behavior due to the confining harmonic potential.



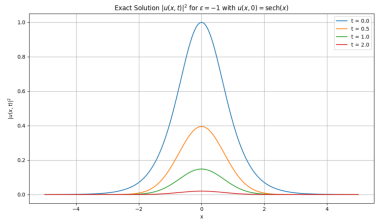
(a) $\varepsilon = \frac{1}{2}$



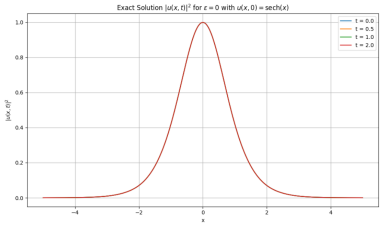
(b) $\varepsilon = 1$



(c) $\varepsilon = 2$



(d) $\varepsilon = -1$



(e) $\varepsilon = 0$

Figure 1: Time evolution of the $|u(x,t)|^2$ with initial condition $u(x,0) = \operatorname{sech}(x)$ and under a harmonic potential.

At positive ε the exact solution via Hermite expansion shows a stable harmonic oscillator and potential well becomes steeper. For moderate values, such as $\varepsilon = 0.5$, 1, the wavefunction remains localized and oscillates periodically. At higher values ($\varepsilon = 2$), oscillations become faster and wavefunction becomes more concentrated around the origin. For $\varepsilon = 0$, it can be seen that wave packet decays as time grows, so the particle is repelled from the origin, leading to delocalization.

3 Chebyshev Spectral Method

The first method that we applied to solve the equation is Chebyshev Spectral Method.

3.1 Algorithm Description

Let $x \in [-10, 10]$. We use Chebyshev-Gauss-Lobatto points (a set of collocation points derived from the extrema of the Chebyshev polynomials of the first kind of degree n to discretize space and apply an explicit time-stepping approach to progress the solution (El-Baghdady&El-Azab, 2016). The steps are:

1. **Selecting parameters:**
   - N, number of Chebyshev points - 1024
   - dt, time step - 0.001
   - T, final time - 0.5
   - $t_{\text{steps}} - T/dt$
2. **Define Chebyshev collocation points:**

$$x_j = L \cos\left(\frac{\pi j}{N-1}\right), \quad j = 0, \ldots, N-1$$

3. **Construct differentiation matrix** D and compute second derivative matrix:

$$D_{ij} = \begin{cases} \frac{(-1)^{i+j}}{x_i - x_j} & i \neq j \\ -\sum_{k \neq i} D_{ik} & i = j \end{cases}, \quad D^{(2)} = D^2$$

4. **Potential:**

$$V(x) = \varepsilon x^2$$

5. **Initial condition:**

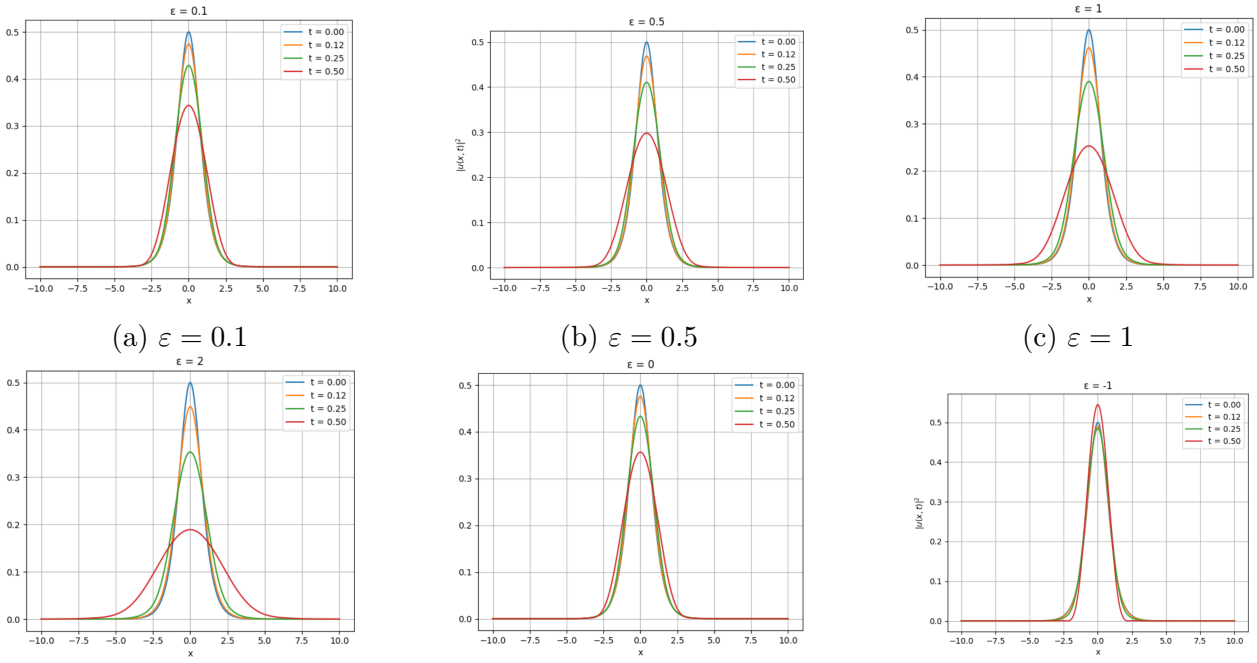
$$u(x, 0) = \operatorname{sech}(x)$$

6. **Time progression (Euler approach):**

$$u^{n+1} = u^n + dt \cdot i \left(D^{(2)} u^n + V u^n\right)$$

3.2 Numerical Results

The following figure shows the evolution of $|u(x,t)|^2$ over time progression, starting from the initial condition $\text{sech}(x)$:



(a) $\varepsilon = 0.1$ (b) $\varepsilon = 0.5$ (c) $\varepsilon = 1$

(d) $\varepsilon = 2$ (e) $\varepsilon = 0$ (f) $\varepsilon = -1$

Figure 2: Evolution over time of $|u(x,t)|^2$ using the Chebyshev method with $u(x,0) = \text{sech}(x)$.

As a result, the Chebyshev spectral method illustrated the stable and accurate performance across all tested values of ε and successfully represents key features of the wavefunction evolution that was seen in the Exact solution. For positive cases $\varepsilon > 0$, the method reproduced the general breathing behavior (wave spreads out and then shrinks back repeatedly over time), with visible amplitude reduction over time. However, for larger values, namely $\varepsilon = 2$, the expected sharpening and strong oscillations were not clearly observed, possibly due to limited time accuracy or numerical smoothing effects. For $\varepsilon = 0$ (free particle case), theoretically, the wavefunction should exhibit smooth dispersive spreading, which was successfully captured. The method showed both symmetry and norm, matching the known analytical behavior. In the repulsive potential case $\varepsilon < 0$, the method remained stable and accurately performed the delocalization as also was observed in exact solution.

4 Fourier Split-Step Spectral Method

The next method with which we solve the Schrödinger equation is Fourier Split-Step Method. The equation is:

$$iu_t = u_{xx} + \varepsilon x^2 u, \quad u(x,0) = \operatorname{sech}(x)$$

on a periodic spatial domain $x \in [-10, 10]$. In this method the kinetic and potential terms are treated separately in Fourier and real space (Bayindir, 2015).

4.1 Algorithm Steps

1. **Discretize the spatial domain** into $N = 256$ grid points:

$$x_j = -L + \frac{2Lj}{N}, \quad j = 0, 1, \ldots, N-1$$

   and compute Fourier wavenumbers:

$$k_n = \frac{2\pi n}{2L}, \quad n = -\frac{N}{2}, \ldots, \frac{N}{2} - 1$$

2. **Initialize the wave function:**

$$u(x,0) = \operatorname{sech}(x)$$

3. **Precompute exponentials:**

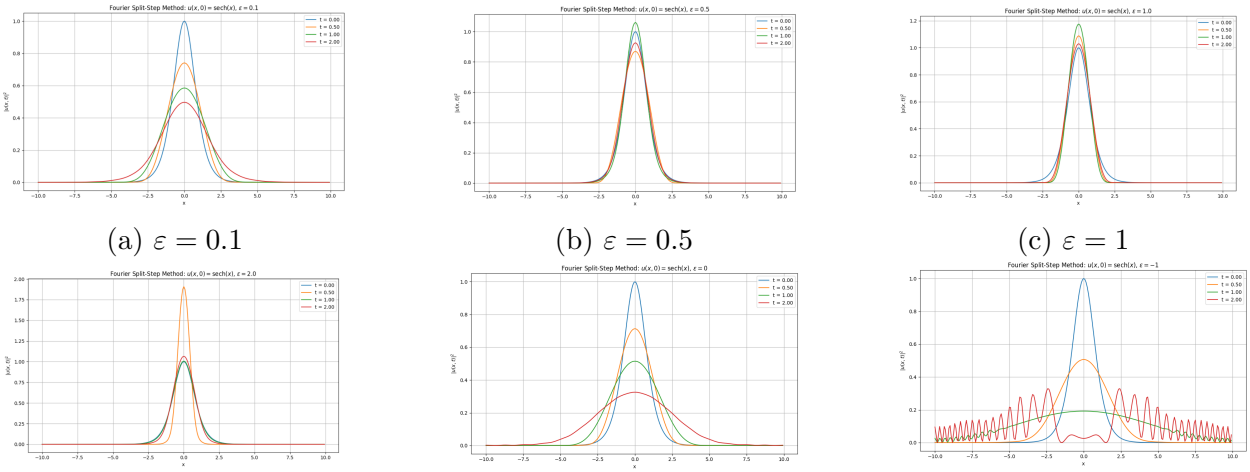
   - $\exp(-ik^2\, dt)$ in Fourier space for the kinetic term
   - $\exp(-i\varepsilon x^2\, dt)$ in real space for the potential term

4. **Time-stepping loop (Split-Step):** For each time step $n = 0, 1, \ldots, T/dt = 2/0.0005$, update the solution:

$$\begin{aligned} \hat{u} &\leftarrow \mathcal{F}[u^n(x)] \\ \hat{u} &\leftarrow \hat{u} \cdot e^{-ik^2 dt} \\ u &\leftarrow \mathcal{F}^{-1}[\hat{u}] \\ u^{n+1}(x) &\leftarrow u(x) \cdot e^{-i\varepsilon x^2 dt} \end{aligned}$$

4.2 Numerical Results

The figures below show the evolution of $|u(x,t)|^2$ over time, starting from the initial condition $\text{sech}(x)$:



(a) $\varepsilon = 0.1$ (b) $\varepsilon = 0.5$ (c) $\varepsilon = 1$

(d) $\varepsilon = 2$ (e) $\varepsilon = 0$ (f) $\varepsilon = -1$

Figure 3: Fourier Split-Step Spectral Method with initial condition $u(x,0) = \text{sech}(x)$.

As can be seen the Fourier spectral method is effective for the free particle case ($\varepsilon = 0$), where it accurately simulated the broadening of the wavepacket up to $t = 1$. For moderate cases $\varepsilon = 0.5$ and $\varepsilon = 1$, the method illustrates the harmonic oscillations, although minor phase inaccuracies became noticeable as time progress. At $\varepsilon = 2$, the periodic boundary conditions inherent in the Fourier basis, as a result of which the method faces the distortion and limitations. Thus, the gradual loss of accuracy can be observed. For negative potential $\varepsilon = -1$, the exponential divergence of the wavepacket led to unphysical interactions with the domain boundaries, what reduce the reliability of the method. Although there are limitations under steep or repulsive potentials, the Fourier spectral method showed computational efficiency and accuracy for all tested values of ε over given time intervals.

5 Crank–Nicolson Method (Sparse Matrix Form)

The next tested method is Crank–Nicolson on the domain $x \in [-10, 10]$, discretized using uniform grid points and finite differences (Kværnø, 2020).

5.1 Algorithm Steps

1. **Discretize the spatial domain:**

$$x_j = -L + j \cdot dx, \quad j = 0, 1, \ldots, N-1$$

where $dx = \frac{2L}{N-1}$ and N = 128.

2. **Define the initial condition:**

$$u(x, 0) = \operatorname{sech}(x)$$

3. **Construct the second derivative operator A:**

$$A = \frac{1}{dx^2} \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{bmatrix}$$

4. **Define the potential term:**

$$V = \operatorname{diag}(\varepsilon x_j^2)$$

5. **Form the Hamiltonian matrix:**

$$H = A + V$$

6. **Construct implicit Crank–Nicolson matrices:**

$$B = I - \frac{idt}{2} H, \quad C = I + \frac{idt}{2} H$$

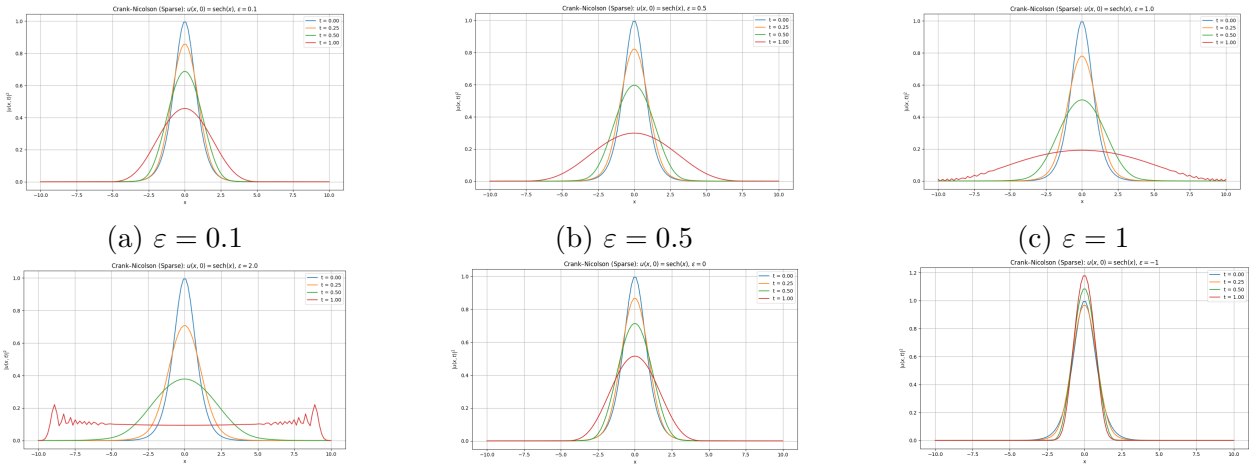
7. **Time-stepping:** At each time step n, solve:

$$Bu^{n+1} = Cu^n$$

using LU (lower-triangular and upper-triangular matrix) decomposition or sparse solvers.

5.2 Numerical Results

The following figure shows the evolution of $|u(x,t)|^2$ over time, starting from the initial condition $\mathrm{sech}(x)$:



(a) $\varepsilon = 0.1$ (b) $\varepsilon = 0.5$ (c) $\varepsilon = 1$

(d) $\varepsilon = 2$ (e) $\varepsilon = 0$ (f) $\varepsilon = -1$

Figure 4: Crank–Nicolson method with initial condition $u(x,0) = \mathrm{sech}(x)$.

As expected, the Crank-Nicolson method showed stable results across different scenarios. In the absence of a potential ($\varepsilon = 0$), it accurately captured the dispersive spreading of the wavefunction while conserving the norm. As epsioln increased from $\varepsilon = 0.5$ to $\varepsilon = 2$, the transition to oscillatory, localized behavior can be seen which is in line with the exact solution. For ($\varepsilon = 0.5$, 1), the wavepacket progress symmetrically. But, at stronger epsilon ($\varepsilon = 2$), we observe amplitude distortion and slight asymmetries. Thus, it can be assumed that the method, although stable, under steep potentials may demonstrate artifacts. In the repulsive case ($\varepsilon = -1$), the exponential spreading is tracked, which indicates that the method remained stable. In general, the qualities of the Crank-Nicolson method, such as implicit formulation and norm preserving properties, make it effective for long time scenarios, but its dynamics under strong confinement should be taken into consideration.

6 Lax–Wendroff Method

The next scheme is Lax–Wendroff method, which is an explicit second-order time integration approach originally applied for hyperbolic partial differential equations. We adapt it here for the Schrödinger equation (Evans,2022).

6.1 Algorithm Steps

1. **Discretize the domain:**

$$x_j = -L + j \cdot dx, \quad j = 0, 1, \ldots, N-1$$

where L=10, N=256, $dx = \frac{2L}{N}$, and we define potential:

$$f_j = \varepsilon x_j^2$$

2. **Initial condition:**

$$u(x, 0) = \operatorname{sech}(x)$$

3. **Finite difference approximations:**

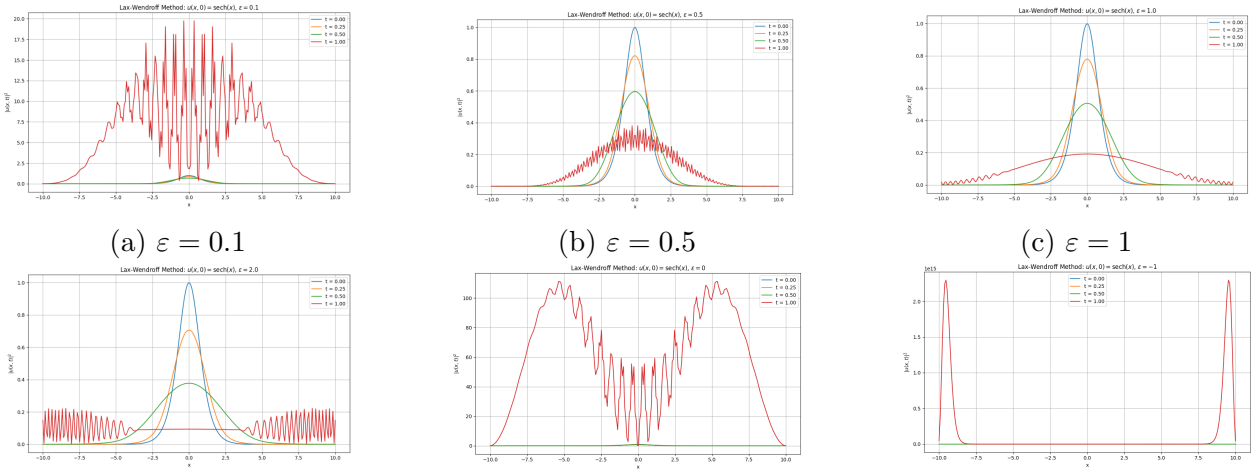
$$\begin{aligned}
(u_{xx})_j^n &\approx \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{dx^2} \\
(u_t)_j^n &= -i\left((u_{xx})_j^n + f_j u_j^n\right) \\
(u_{tt})_j^n &= -i\left(\frac{(u_t)_{j+1}^n - 2(u_t)_j^n + (u_t)_{j-1}^n}{dx^2} + f_j (u_t)_j^n\right)
\end{aligned}$$

4. **Time stepping update:**

$$u_j^{n+1} = u_j^n + dt \cdot (u_t)_j^n + \frac{dt^2}{2} \cdot (u_{tt})_j^n$$

6.2 Numerical Results

The following graphs shows the evolution of $|u(x,t)|^2$ over time, starting from the initial condition $\operatorname{sech}(x)$:



(a) $\varepsilon = 0.1$ (b) $\varepsilon = 0.5$ (c) $\varepsilon = 1$

(d) $\varepsilon = 2$ (e) $\varepsilon = 0$ (f) $\varepsilon = -1$

Figure 5: Lax–Wendroff method with initial condition $u(x, 0) = \text{sech}(x)$.

So, the method produced sufficiently acceptable results for the free particle case at early times, but its limitations quickly became evident as both simulation time and potential strength increased. For small confinement ($\varepsilon = 0.5$), wave dispersion started to produce numerical noise. At $\varepsilon = 1$, visible phase distortion appeared as time develops by $t = 0.5$, and by $t = 1.0$, so the solution was significantly degraded. Under strong potential ($\varepsilon = 2$) and especially in the inverted potential ($\varepsilon = -1$), the method became unstable and yielded unreliable results. Overall, although the method is second order accurate, fully explicit and easy to implement, it is not norm preserving and can become unstable for large ε or time steps. Thus, it is better to apply this method for weakly nonlinear or linear problems on fine grids with smaller time steps.

7 Leapfrog Method

The next method, which is similar to previous one, is Leapfrog method. This approach is also an explicit second order time integration scheme, usually used for wave like partial differential equations (Evans, 2022).

7.1 Algorithm Steps

1. **Discretize the spatial domain:**

$$x_j = -L + j \cdot dx, \quad j = 0, \ldots, N-1, \quad dx = \frac{2L}{N}$$

where L=10, N=256

2. **Define the potential:**

$$f_j = \varepsilon x_j^2$$

3. **Initialize:**

$$u_j^0 = \operatorname{sech}(x_j)$$

(First time step using Euler method):

$$u_j^1 = u_j^0 + dt \cdot \left(-i \left(\frac{u_{j+1}^0 - 2u_j^0 + u_{j-1}^0}{dx^2} + f_j u_j^0 \right) \right)$$

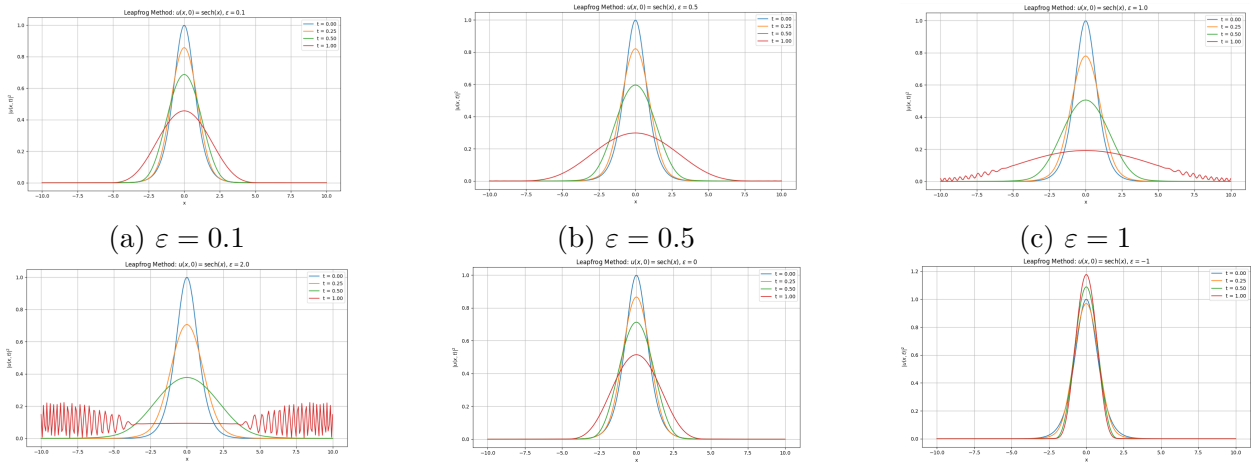
4. **Leapfrog update (for $n \geq 1$):**

$$u_j^{n+1} = u_j^{n-1} + 2dt \cdot \left(-i \left(\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{dx^2} + f_j u_j^n \right) \right)$$

5. **Repeat** for $n = 1, \ldots, T/dt$, and compute $|u(x,t)|^2$. The T=1, dt=0.001

7.2 Numerical Results

The following figures present the evolution of $|u(x,t)|^2$ over time, starting from the initial condition $\operatorname{sech}(x)$:



(a) $\varepsilon = 0.1$ (b) $\varepsilon = 0.5$ (c) $\varepsilon = 1$

(d) $\varepsilon = 2$ (e) $\varepsilon = 0$ (f) $\varepsilon = -1$

Figure 6: Leapfrog method for the Schrödinger equation with initial condition $u(x,0) = \operatorname{sech}(x)$.

As can be seen from the figure, the Leapfrog method exhibited mixed performance depending on the strength of the potential. For small values of ε, the method was efficient

and accurate over short simulation times, which is similar to exact solution. As ε increased, the method became more sensitive to the time-step size and showed dispersion. In contrast, in the repulsive potential case ($\varepsilon = -1$), the method provided good results. It successfully presented the symmetric delocalization and outward acceleration of the wavepacket. Thus, it can be assumed that Leapfrog scheme can remain reliable when the system is spreading outward, especially for smooth initial conditions (as in our case $\text{sech}(x)$ and smaller time steps.

8 Comparison of Numerical Methods

In the table below we summarized the results of all tested numerical methods for solving Schrödinger equation.

For each method we considered its accuracy, stability, and time efficiency across different ε values and time up to $t = 2$.

Method	**Accuracy**	**Stability**	**Time Efficiency**
Exact Solution	Benchmark, as it provides true physical behavior.	Perfect.	Fast.
Chebyshev Method	Very accurate for smooth solutions; less accurate at steep gradients.	Stable, slight stiffness at larger ε.	Moderate (due to matrix operations).
Fourier Method	Accurate, but resolves oscillations.	Stable; reduced reliability at longer time.	Fast (due to FFT base).
Crank–Nicolson	Norm preserving; slight fluctuations at higher ε.	Stable, but exhibit small oscillations at steep potentials.	Moderate (due to sparse solvers).
Lax–Wendroff	Poor for almost all ε; instability grows as time increases.	Unstable.	Fast.
Leapfrog	Good at $\varepsilon \leq 1$, but loses accuracy after.	Oscillations at high ε.	Efficient but needs smaller time steps.

- **Best method considering all factors:**
  Crank–Nicolson and Chebyshev Method
- **Best for speed:**
  Fourier and Leapfrog Method
- **Worst perfomance:**
  Lax–Wendroff Method

9 Duhamel's Principle

In addition to classical numerical methods, we studied the Schrödinger equation analytically by Duhamel's principle:

9.1 Algorithm steps

Step 1: The equation can be written as:

$$iu_t = \underbrace{u_{xx}}_{\text{linear evolution}} + \underbrace{\varepsilon x^2 u}_{\text{non-constant coefficient term}}.$$

We can consider it as:

$$iu_t = L_0 u + V(x)u, \quad \text{where } L_0 = \partial_{xx}, \quad V(x) = \varepsilon x^2.$$

Step 2: Homogeneous solution (known as free Schrödinger equation)

Firstly, we consider the Schrödinger equation without potential:

$$iu_t^{(0)} = u_{xx}^{(0)}, \quad u^{(0)}(x,0) = u_0(x).$$

The solution to this equation is given by convolution with the Schrödinger kernel (Różański, 2021) :

$$u^{(0)}(x,t) = (G(t) * u_0)(x),$$

where the Green's function is:

$$G(x,t) = \frac{1}{\sqrt{4\pi i t}} \exp\left(\frac{ix^2}{4t}\right).$$

Thus, the solution is:

$$u^{(0)}(x,t) = \int_{-\infty}^{\infty} G(x-y,t)u_0(y)\,dy.$$

Step 3: Applying Duhamel's principle

It is known that according to Duhamel's principle the solution to the inhomogeneous equation:

$$iu_t = u_{xx} + \varepsilon x^2 u, \quad u(x,0) = u_0(x),$$

can be written as:

$$u(x,t) = u^{(0)}(x,t) - i\int_0^t G(t-s) * [\varepsilon x^2 u(x,s)]\, ds.$$

So, the final result is:

$$u(x,t) = \int_{-\infty}^{\infty} G(x-y,t)u_0(y)\, dy - i\varepsilon \int_0^t \int_{-\infty}^{\infty} G(x-y,t-s)\cdot y^2 u(y,s)\, dy\, ds.$$

where since $x^2u(x,s)$ is under convolution, the variable is changed to y for integration.

9.2 Numerical evaluation for initial condition sech(x)

We will use the perturbative expansion of the solution u:

$$u = u^{(0)} + \varepsilon u^{(1)} + \varepsilon^2 u^{(2)} + \cdots$$

Zeroth-order solution $u^{(0)}$

We solve:

$$i\partial_t u^{(0)} = \partial_{xx} u^{(0)}, \quad u^{(0)}(x,0) = u_0(x)$$

The solution is given by convolution with the Green's function (Różański, 2021):

$$u^{(0)}(x,t) = \int_{-\infty}^{\infty} G(x-y,t)u_0(y)\, dy$$

where

$$G(x,t) = \frac{1}{\sqrt{4\pi i t}} \exp\left(\frac{ix^2}{4t}\right)$$

First-order correction $u^{(1)}$

We substitute into the original equation and unite $\mathcal{O}(\varepsilon)$ terms:

$$i\partial_t u^{(1)} = \partial_{xx} u^{(1)} + x^2 u^{(0)}, \quad u^{(1)}(x,0) = 0$$

Applying the Duhamel's principle, the first-order correction is:

$$u^{(1)}(x,t) = -i\int_0^t \int_{-\infty}^{\infty} G(x-y,t-s)\cdot y^2 u^{(0)}(y,s)\, dy\, ds$$

$$u(x,t) = u^{(0)}(x,t) + \varepsilon u^{(1)}(x,t) + \mathcal{O}(\varepsilon^2)$$

Let us consider an initial condition:

$$u_0(x) = sech(x)$$

So the free Schrödinger solution is known: $\mathrm{u}^{(0)}(x,t) = \int_{-\infty}^{\infty} G(x-y,t)\,\mathrm{sech}(y)\,dy$,

$$u(x,t) \approx u^{(0)}(x,t) - i\varepsilon \int_0^t \int_{-\infty}^{\infty} G(x-y,t-s)\cdot y^2 u^{(0)}(y,s)\,dy\,ds$$

We then define the first-order correction as:

$$u^{(1)}(x,t) = -i \int_0^t \int_{-\infty}^{\infty} G(x-y,t-s)\cdot y^2 u^{(0)}(y,s)\,dy\,ds$$

Thus, for each ε, we calculate the approximation:

$$u(x,t) \approx u^{(0)}(x,t) + \varepsilon u^{(1)}(x,t)$$

9.3 Behavior for positive ε

We analyze the behavior of the solution $|u(x,t=1)|$ firstly under the positive values of the parameter ε from the set:

$$\varepsilon \in \left\{0, \frac{1}{100}, \frac{1}{10}, 1\right\}$$

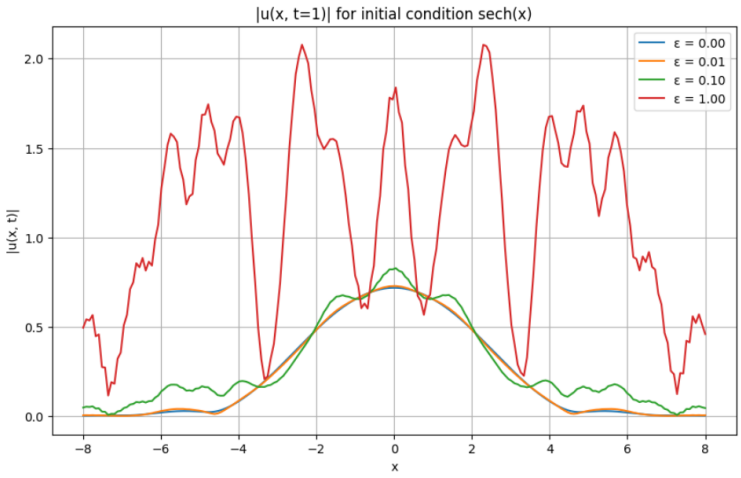


Figure 7: Solution of the Schrödinger equation using Duhamel's principle with $\varepsilon \geq 0$

As ε increases, the wavefunction shows oscillations and peak sharpening. This effect is more noticeable as $\varepsilon = 1$, indicating potential sensitivity to the spectral aliasing arising from the integral term or time stepping.

9.4 Behavior for negative ε

Then, we analyze the behavior of the solution under the negative values of the parameter ε from the set:

$$\varepsilon \in \left\{0, -\frac{1}{100}, -\frac{1}{10}, -1\right\}$$

These values indicate the repulsive quadratic potential of the form:

$$-|\varepsilon|x^2 u,$$

which presents an inverted harmonic potential, pushing the wavepacket away from the origin.

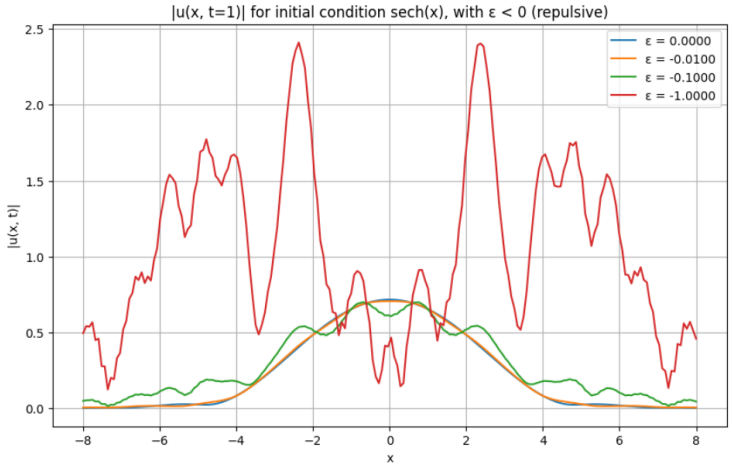


Figure 8: Numerical solution of the Schrödinger equation using Duhamel's principle with $\varepsilon < 0$

For negative cases$\varepsilon < 0$, the wavefunction shows more noticeable spreading due to the repulsive potential. At $\varepsilon = -1$, as in the positive potential case, this effect becomes stronger and resulted in visible peak splitting and the formation of oscillatory tails. So, for both cases the principle shows signs of instability when $|\varepsilon|$ becomes large, especially by the time $t = 1$.

10 Time discretization method

Another analytical solution to study is the time-splitting method, which solves the equation sequentially over small time steps:

$$\begin{aligned}[t_0, t_1]: &\quad iu_t = u_{xx} \\ [t_1, t_2]: &\quad iu_t = \varepsilon x^2 u\end{aligned}$$

We repeat this process until $t = T$.

Step 1: Solving $iu_t = u_{xx}$

This equation can be solved in Fourier space (Ascher, 1995).

Let $\hat{u}(k,t)$ be the Fourier transform of $u(x,t)$. Then:

$$i\partial_t \hat{u}(k,t) = -k^2 \hat{u}(k,t) \qquad \hat{u}(k,t) = \hat{u}(k,0)e^{ik^2 t}$$

Then we invert the Fourier transform to get $u(x,t)$.

Given initial condition:

$$u(x,0) = \operatorname{sech}(x) \qquad \hat{u}(k,0) = \pi \cdot \operatorname{sech}\left(\frac{\pi k}{2}\right)$$

So, the final result is:

$$u(x,t_1) = \mathcal{F}^{-1}\left[\hat{u}(k,0) \cdot e^{ik^2 t_1}\right](x)$$

Step 2: Solving $iu_t = \varepsilon x^2 u$

This can be recognized as a point wise phase shift:

$$u(x,t) = u(x,t_1) \cdot \exp(-i\varepsilon x^2 (t - t_1))$$

Thus, at t_2:

$$u(x,t_2) = u(x,t_1) \cdot \exp(-i\varepsilon x^2 \Delta t)$$

Summary of the process

In general, for each time interval $[t_n, t_{n+1}]$:

1. Solve $iu_t = u_{xx}$ by FFT (Fast Fourier Transformation):

$$\hat{u}(k, t_{n+1}) = \hat{u}(k, t_n) e^{ik^2 \Delta t}$$

2. Solve $iu_t = \varepsilon x^2 u$ pointwise:

$$u(x, t_{n+1}) = u(x, t_{n+1}) \cdot e^{-i\varepsilon x^2 \Delta t}$$

Graphical Representation

The figure below shows the evolution of wavefunction spreading using the time discretization method.

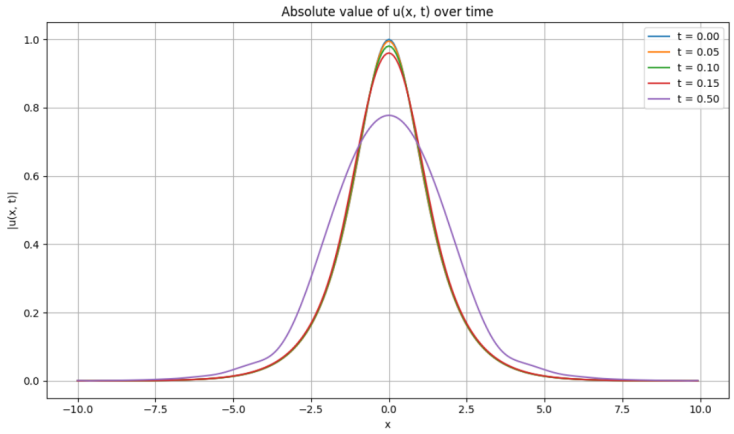


Figure 9: Time Discretization method with initial condition $\mathrm{sech}(x)$

This method is efficient and accurately captures the exact solution features. Its accuracy may be reduced for large potentials unless symmetric splitting or smaller time steps are used.

11 Modified Potential

For deeper understanding of the effect of external potentials, we shift focus to the Schrödinger equation with a modified potential of the form:

$$V(x) = |x|^2 - \varepsilon |x|^b, \quad \text{where} \quad 0 < b < 2 \quad \text{and} \quad \varepsilon > 0.$$

This potential indicates a perturbed trapping well since it behaves harmonically near the origin but is weakened further out, depending on the parameters ε and b. In contrast to the harmonic oscillator potential, that we previously discussed, the exact Hermite spectral

solutions are no longer applicable. However, we can approach the problem using perturbation theory and approximate it as an exact solution.

11.1 Exact solution (approximated by perturbation theory)

Firstly, we define the unperturbed basis functions $\varphi_n(x)$ as the normalized Hermite functions:

$$\varphi_n(x) = \frac{1}{\sqrt{2^n n! \sqrt{\pi}}} H_n(x) e^{-x^2/2},$$

where $H_n(x)$ is the n-th Hermite polynomial.
Secondly, we consider the first order correction to the energy eigenvalues, which is given by:

$$E_n^{(1)} = -\int_{-\infty}^{\infty} |\varphi_n(x)|^2 |x|^b \, dx.$$

Since this integral can't be solved analytically for $b \in (0, 2)$, we may compute it numerically.
Thirdly, assuming the initial condition is $u_0(x) = \operatorname{sech}(x)$, we project it onto the Hermite basis:

$$c_n = \int_{-\infty}^{\infty} u_0(x) \varphi_n(x) \, dx.$$

Fourthly, the solution is then approximated by:

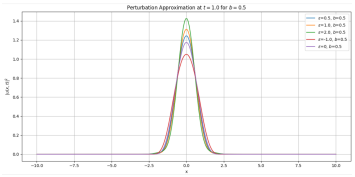
$$u(x,t) = \sum_n c_n \varphi_n(x) \exp\left[-i\left(E_n^{(0)} + \varepsilon E_n^{(1)}\right) t\right],$$

where $E_n^{(0)} = 2n+1$ are the unperturbed energy levels of the harmonic oscillator (Grillakis, 2020).

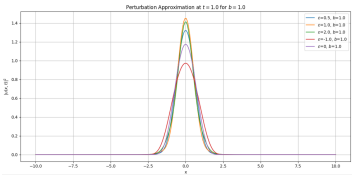
11.2 Graphical Results

The graphs below show the perturbation theory approximation of the equation under the potential:

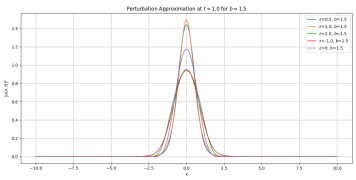
$$V(x) = |x|^2 - \varepsilon |x|^b, \quad \text{where} \quad 0 < b < 2 \quad \text{and} \quad \varepsilon > 0.$$



(a) $\varepsilon = 0.5, 1, 2, 0, -1; b = 0.5$



(b) $\varepsilon = 0.5, 1, 2, 0, -1; b = 1$



(c) $\varepsilon = 0.5, 1, 2, 0, -1; b = 1.5$

Figure 10: Exact solution (perturbation method) with initial condition $u(x, 0) = \text{sech}(x)$.

The numerical results illustrates that when $\varepsilon = 0$, the dynamic is similar to the standard harmonic oscillator, performing a localized and symmetric wavefunction, as expected. For $\varepsilon > 0$, the subtractive term reduces the confinement, which leads to slight expansion of the wavepacket. As ε and b increases, the effect becomes more visible. However, the shape stays largely bounded and symmetric due to the stronger influence of the quadratic term. When $\varepsilon = -1$, the perturbation becomes additive, making the well deeper. As a result, a visibly sharper and more localized wavefunction can be seen, especially for larger b.

12 Chebyshev method results with modified potential

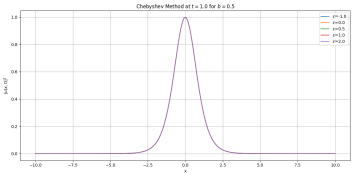
The figures below show the solution to the Schrödinger equation at $t = 1.0$, using the Chebyshev Spectral Method for the modified potential:

$$V(x) = |x|^2 - \varepsilon |x|^b,$$

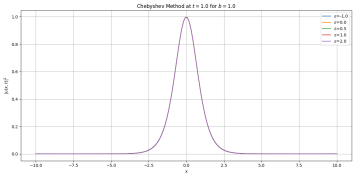
with values ranges:

$$\varepsilon \in \{-1,\ 0,\ 0.5,\ 1.0,\ 2.0\}, \quad b \in \{0.5,\ 1.0,\ 1.5\}.$$

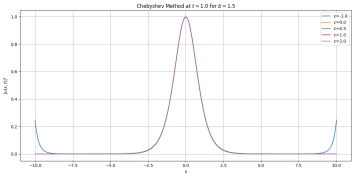
Graphical results



(a) $\varepsilon = 0.5, 1, 2, 0, -1; b = 0.5$



(b) $\varepsilon = 0.5, 1, 2, 0, -1; b = 1$



(c) $\varepsilon = 0.5, 1, 2, 0, -1; b = 1.5$

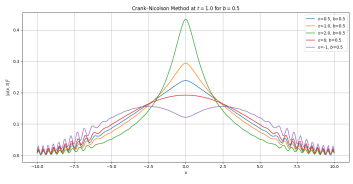
Figure 11: Chebyshev method with initial condition $u(x, 0) = \text{sech}(x)$ and modified potential.

Numerical implementation showed that at the time $t = 1.0$, the Chebyshev method produce static wavefunctions across all tested range of parameters. This insensitivity to the difference variations in the potential may signal about the strong trapping effect of the leading quadratic term $|x|^2$ and short time stimulation. Even for negative ε, the potential remains overall confining near the origin, and the wavefunction does not change within this timeframe. As a result, in this method, the effects of the modified potential are not yet dynamically significant.

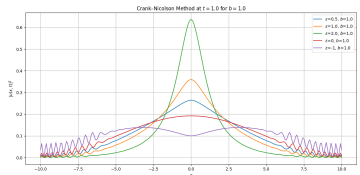
13 Crank–Nicolson method results with modified potential

The three plots below show the solution to the Schrödinger equation at $t = 1.0$, computed using the Crank-Nicolson Method.

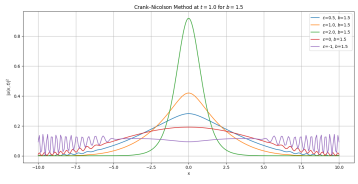
Graphical results



(a) $\varepsilon = 0.5, 1, 2, 0, -1; b = 0.5$



(b) $\varepsilon = 0.5, 1, 2, 0, -1; b = 1$



(c) $\varepsilon = 0.5, 1, 2, 0, -1; b = 1.5$

Figure 12: Crank-Nicolson method with initial condition $u(x, 0) = \text{sech}(x)$ and modified potential.

In the Crank-Nicolson method simulations, as ε increases from -1 to 2, the wavepacket broadens largely and the peak amplitude decreases, which reflects the subtractive term $-\varepsilon|x|^b$ with its weakening of the confining potential effect. When $\varepsilon = -1$, the wavefunction is more localized, with a sharp central peak and steep decay. This indicates that the potential well is successfully deepened. At the same time, for stronger potential $\varepsilon = 2$, the wavepacket performs reduced central density and good spreading, especially with $b = 1.5$. Unlike the Chebyshev outputs, the results of Crank-Nicolson method shows visibly diverge across parameter sets, confirming the significant effect of both ε and b in shaping quantum evolution under modified conditions.

14 Comparison of the methods for modified potential

Below is a summary of the three methods applied for the modified potential - perturbation theory, the Chebyshev method, and the Crank-Nicolson scheme.

$$with \quad \varepsilon \in \{-1,\, 0,\, 0.5,\, 1.0,\, 2.0\}, \quad b \in \{0.5,\, 1.0,\, 1.5\}.$$

The evaluation focuses on the method's accuracy, sensitivity to potential parameters, stability and time efficiency.

Method	**Accuracy**	**Sensitivity to ε, b**	**Stability**	**Cost Efficiency**	**Comments**
Perturbation Theory	High	Moderate sensitivity to ε, b	Very stable	Very high	Fast and analytically approximated to exact solution, but diverges for large ε or long time
Chebyshev Method	High	Low visual sensitivity	Stable	High	Successful results, but effects of potential are small for short time
Crank-Nicolson Method	High	High, since there is clear visual divergence across different potential cases	Stable	Moderate	Unconditionally stable

Table 1: Comparison of numerical methods for solving the Schrödinger equation with modified potential $V(x) = |x|^2 - \varepsilon|x|^b$.

15 Conclusion

In this project, we performed a detailed numerical analysis of the linear Schrödinger equation with external potentials, such as the quadratic form $V(x) = \varepsilon x^2$ and the modified version $V(x) = |x|^2 - \varepsilon|x|^b$, with $0 < b < 2$. Our goal was to gain insights how the strength and shape of the external potential influence the solution behavior and to evaluate the performance of numerical methods. For the quadratic potential, it can be concluded that most methods perform well for small ε, but Chebyshev and Crank–Nicolson method is highlighted as they maintain accuracy and stability as ε increases. Explicit approaches, such as Lax–Wendroff and Leapfrog, although cost efficient, showed dispersion and norm loss under strong potential. At the same time, the Fourier method remained efficient but sensitive to phase distortion. The Hermite exact solution served as a benchmark, and the Duhamel approach showed mixed perfomance and was computationally demanding.

To deeper analyze the effect of the potential, we analyzed the extended version $V(x) = |x|^2 - \varepsilon|x|^b$. In this case we observed that Crank–Nicolson method captured the effects of varying ε and b, showing wavefunction spreading for large ε and sharpening for $\varepsilon < 0$. Perturbation theory provided fast and accurate estimates, while Chebyshev method remained accurate, but static, not showing sensitivity to changes in potentials.

Overall, our findings show that external potential, indeed, has a significant impact on shaping quantum dynamics. The Crank–Nicolson and Chebyshev methods are most effective. For weaker potentials or faster results, Fourier and Leapfrog may be used. Method selection should balance the potential strength, desired accuracy and computational efficiency. Further investigation should be conducted considering other initial conditions, more complex frameworks and non-linear conditions.

References

[1] Ascher, U. M., Ruuth, S. J., & Wetton, B. T. R. (1995). Implicit-explicit methods for time-dependent PDEs. *SIAM Journal on Numerical Analysis*, **32**(3), 797-823.

[2] Bayindir, C. (2015). Compressive split-step Fourier method. *TWMS Journal of Applied and Engineering Mathematics*, **5**(2), 298-306.

[3] Dattoli, G., Chiccoli, C., Lorenzutta, S., Maino, G., & Torre, A. (1994). Theory of generalized Hermite polynomials. *Computers & Mathematics with Applications*, **28**(4), 71-83.

[4] El-Baghdady, G. I., & El-Azab, M. S. (2016). Chebyshev-Gauss-Lobatto Pseudo-spectral Method for One-Dimensional Advection-Diffusion Equation with Variable Coefficients. *Sohag Journal of Mathematics*, **3**(1), 7-14

[5] Evans, L. C. (2022). *Partial differential equations* (Vol. 19). American Mathematical Society.

[6] Feit, M. D., Fleck Jr, J. A., & Steiger, A. (1982). Solution of the Schrödinger equation by a spectral method. *Journal of Computational Physics*, **47**(3), 412-433.

[7] Grillakis, M. G. (2000). On nonlinear Schrödinger equations: Nonlinear Schrödinger equations. *Communications in Partial Differential Equations*, **25**(9-10), 1827-1844.

[8] Kværnø, A. (2020). *Partial differential equations and finite difference methods*. Lecture notes, Nov 4, 2020.

[9] Różański, M., Sikora, B., Smuda, A., and Witula, R. (2021). On theoretical and practical aspects of Duhamel's integral. *Archives of Control Sciences*, **31**(LXVII), no.4, pp.815-847.

16 Appendix

Listing 1: Exact solution (Hermite)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import eval_hermite, factorial
from scipy.integrate import quad

# Hermite function
def hermite_function(n, x, epsilon):
    if epsilon <= 0:
        # Use fallback xi = x if epsilon is 0 or negative
        xi = x
    else:
        xi = epsilon**0.25 * x
    Hn = eval_hermite(n, xi)
    normalization = 1.0 / np.sqrt(np.sqrt(np.pi) * 2**n * factorial(n))
    return normalization * Hn * np.exp(-0.5 * xi**2)

# Compute c_n projection coefficient
def compute_cn(n, u0_func, epsilon):
    integrand = lambda x: u0_func(x) * hermite_function(n, x, epsilon)
    result, _ = quad(integrand, -10, 10)
    return result

def exact_solution(x, t, epsilon, N_terms, u0_func):
    if epsilon == 0:
        sqrt_e = 1e-8
    elif epsilon < 0:
        sqrt_e = np.sqrt(-epsilon) * 1j
    else:
        sqrt_e = np.sqrt(epsilon)

    result = np.zeros_like(x, dtype=complex)
    for n in range(N_terms):
        cn = compute_cn(n, u0_func, epsilon)
```

```
        phi_n = hermite_function(n, x, epsilon)
        E_n = (2 * n + 1) * sqrt_e
        result += cn * phi_n * np.exp(1j * E_n * t)
    return result


u0_func = lambda x: 1 / np.cosh(x)


epsilon_values = [2.0, 1.0, 0.5, 0, -1]
x_vals = np.linspace(-5, 5, 400)
t_vals = [0, 0.5, 1.0, 2.0]
N_terms = 20

for epsilon in epsilon_values:
    plt.figure(figsize=(10, 6))
    for t in t_vals:
        try:
            u = exact_solution(x_vals, t, epsilon, N_terms, u0_func)
            plt.plot(x_vals, np.abs(u)**2, label=f"t = {t:.1f}")
        except Exception as e:
            print(f"Failed for   ={epsilon}, t={t}: {e}")
            continue
    plt.title(rf"Exact Solution $|u(x,t)|^2$ for $\varepsilon = {epsilon}$ wi
    plt.xlabel("x")
    plt.ylabel(r"$|u(x,t)|^2$")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

Listing 2: Fourier Method

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, ifft
```

```
L = 10
N = 256
x = np.linspace(-L, L, N, endpoint=False)
dx = x[1] - x[0]
k = 2 * np.pi * np.fft.fftfreq(N, d=dx)
t_max = 2.0
dt = 0.0005
t_steps = int(t_max / dt)


u0 = 1 / np.cosh(x)

# General Fourier split-step method
def fourier_split_step(u0, f, L, t_steps, dt):
    dx = 2 * L / N
    k = 2 * np.pi * np.fft.fftfreq(N, d=dx)
    u = u0.copy().astype(np.complex128)
    u_history = [u.copy()]

    exp_k = np.exp(-1j * k**2 * dt)
    exp_f = np.exp(-1j * f * dt)

    for _ in range(t_steps):
        u_hat = fft(u)
        u_hat *= exp_k
        u = ifft(u_hat)
        u *= exp_f
        u_history.append(u.copy())

    return np.array(u_history), x


epsilons = [0.1, 0.5, 1.0, 2.0, 0, -1]
f_base = x**2
time_indices = [0, int(0.5 / dt), int(1.0 / dt), int(2.0 / dt)]
```

```
for eps in epsilons:
    f = eps * f_base
    u_hist, x_vals = fourier_split_step(u0, f, L, t_steps, dt)
    plt.figure(figsize=(10, 6))
    for idx in time_indices:
        plt.plot(x_vals, np.abs(u_hist[idx])**2, label=f"t = {idx * dt:.2f}")
    plt.title(f"Fourier Split-Step Method: $u(x,0) = \\mathrm{{sech}}(x)$, $\
    plt.xlabel("x")
    plt.ylabel(r"$|u(x,t)|^2$")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

Listing 3: Crank-Nicolson

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse import diags, identity
from scipy.sparse.linalg import splu

L = 10
N = 128
x = np.linspace(-L, L, N)
dx = x[1] - x[0]

t_max = 1.0
dt = 0.001
t_steps = int(t_max / dt)
time_indices = [0, int(0.25 / dt), int(0.5 / dt), int(1.0 / dt)]

u0 = 1 / np.cosh(x)
f_base = x**2

# Crank Nicolson method
```

```
def crank_nicholson_sparse(u0, f, dx, dt, t_steps):
    u = u0.copy().astype(np.complex128)
    u_history = [u.copy()]

    # Discrete Laplacian using central differences
    main_diag = -2 * np.ones(N) / dx**2
    off_diag = np.ones(N - 1) / dx**2
    laplacian = diags([off_diag, main_diag, off_diag], offsets=[-1, 0, 1], fo

    V = diags(f, 0, format='csc')
    H = laplacian + V

    I = identity(N, format='csc', dtype=np.complex128)
    B = (I - 0.5j * dt * H)
    C = (I + 0.5j * dt * H)

    B_lu = splu(B)

    for _ in range(t_steps):
        u = B_lu.solve(C @ u)
        u_history.append(u.copy())

    return np.array(u_history), x


epsilons = [0.1, 0.5, 1.0, 2.0, 0,-1]

for eps in epsilons:
    f = eps * f_base
    u_hist, x_vals = crank_nicholson_sparse(u0, f, dx, dt, t_steps)

    plt.figure(figsize=(10, 6))
    for idx in time_indices:
        plt.plot(x_vals, np.abs(u_hist[idx])**2, label=f"t = {idx * dt:.2f}")

    plt.title(f" C r a n k Nicolson  (Sparse): $u(x,0) = \\mathrm{{sech}}(x)$, $\
```

```
    plt.xlabel("x")
    plt.ylabel(r"$|u(x,t)|^2$")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

Listing 4: Lax-Wendroff Method

```
import numpy as np
import matplotlib.pyplot as plt



L = 10
N = 256
x = np.linspace(-L, L, N)
dx = x[1] - x[0]

t_max = 1.0
dt = 0.001
t_steps = int(t_max / dt)
time_indices = [0, int(0.25 / dt), int(0.5 / dt), int(1.0 / dt)]



u0 = 1 / np.cosh(x)

# Lax-Wendroff method
def lax_wendroff_method(u0, f, dx, dt, t_steps):
    u = u0.copy().astype(np.complex128)
    u_history = [u.copy()]

    for _ in range(t_steps):
        u_plus = np.roll(u, -1)
        u_minus = np.roll(u, 1)
        u_xx = (u_plus - 2 * u + u_minus) / dx**2
        u_t = -1j * (u_xx + f * u)
        u_tt = -1j * ((np.roll(u_t, -1) - 2 * u_t + np.roll(u_t, 1)) / dx**2
```

```
        u = u + dt * u_t + 0.5 * dt**2 * u_tt
        u_history.append(u.copy())

    return np.array(u_history), x


epsilons = [0.1, 0.5, 1.0, 2.0, 0, -1]
f_base = x**2

for eps in epsilons:
    f = eps * f_base
    u_hist, x_vals = lax_wendroff_method(u0, f, dx, dt, t_steps)
    plt.figure(figsize=(10, 6))
    for idx in time_indices:
        plt.plot(x_vals, np.abs(u_hist[idx])**2, label=f"t = {idx * dt:.2f}")
    plt.title(f"Lax–Wendroff Method: $u(x,0) = \\mathrm{{sech}}(x)$, $\\varep
    plt.xlabel("x")
    plt.ylabel(r"$|u(x,t)|^2$")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

Listing 5: Leapfrog Method

```
import numpy as np
import matplotlib.pyplot as plt


L = 10
N = 256
x = np.linspace(-L, L, N)
dx = x[1] - x[0]

t_max = 1.0
dt = 0.001
t_steps = int(t_max / dt)
```

```
time_indices = [0, int(0.25 / dt), int(0.5 / dt), int(1.0 / dt)]


u0 = 1 / np.cosh(x)

# Leapfrog method
def leapfrog_method(u0, f, dx, dt, t_steps):
    u = u0.copy().astype(np.complex128)
    u_prev = u.copy()

    # First step using forward Euler
    u_plus = np.roll(u, -1)
    u_minus = np.roll(u, 1)
    u_xx = (u_plus - 2 * u + u_minus) / dx**2
    u1 = u + dt * (-1j * (u_xx + f * u))

    u_history = [u.copy(), u1.copy()]

    # Leapfrog iteration
    for _ in range(2, t_steps + 1):
        u_plus = np.roll(u1, -1)
        u_minus = np.roll(u1, 1)
        u_xx = (u_plus - 2 * u1 + u_minus) / dx**2
        u_next = u_prev + 2 * dt * (-1j * (u_xx + f * u1))
        u_prev, u1 = u1, u_next
        u_history.append(u1.copy())

    return np.array(u_history), x


epsilons = [0.1, 0.5, 1.0, 2.0, 0, -1]
f_base = x**2


for eps in epsilons:
    f = eps * f_base
```

```
    u_hist, x_vals = leapfrog_method(u0, f, dx, dt, t_steps)
    plt.figure(figsize=(10, 6))
    for idx in time_indices:
        plt.plot(x_vals, np.abs(u_hist[idx])**2, label=f"t = {idx * dt:.2f}")
    plt.title(f"Leapfrog Method: $u(x,0) = \\mathrm{{sech}}(x)$, $\\varepsilo
    plt.xlabel("x")
    plt.ylabel(r"$|u(x,t)|^2$")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

Listing 6: Chebyshev Method

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.polynomial.chebyshev import chebpts1

L = 10
N = 1024
t_max = 0.5
dt = 0.001
t_steps = int(t_max / dt)

# Generalized Chebyshev Method Function with epsilon parameter
def chebyshev_method(u0_func, epsilon, L, N, t_steps, dt):
    x_cheb = L * chebpts1(N)
    u = u0_func(x_cheb)
    V = epsilon * x_cheb**2

    # Construct Chebyshev differentiation matrix D
    D = np.zeros((N, N))
    for i in range(N):
        for j in range(N):
            if i != j:
                D[i, j] = (-1)**(i + j) / (x_cheb[i] - x_cheb[j])
        D[i, i] = -np.sum(D[i, :])
```

```
    D2 = D @ D
    D2 = D2 / np.max(np.abs(D2))

    u_history = [u.copy()]
    for _ in range(t_steps):
        u = u + dt * 1j * (D2 @ u + V * u)
        u_history.append(u.copy())
    return np.array(u_history), x_cheb

u0_func = lambda x: 1 / np.cosh(x)


epsilons = [0.1, 0.5, 1.0, 2.0, 0, -1]
time_indices = [0, int(0.5 / dt), int(1.0 / dt), int(2.0 / dt)]


for eps in epsilons:
    u_hist, x_cheb = chebyshev_method(u0_func, eps, L, N, t_steps, dt)
    plt.figure(figsize=(10, 6))
    for idx in time_indices:
        plt.plot(x_cheb, np.abs(u_hist[idx])**2, label=f"t = {idx * dt:.2f}")
    plt.xlabel("x")
    plt.ylabel(r"$|u(x,t)|^2$")
    plt.title(f"Chebyshev Method: $u(x,0) = \\mathrm{{sech}}(x)$, $\\varepsil
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

Listing 7: Chebyshev Method with modified potential

```
from numpy.polynomial.chebyshev import chebpts1

# Chebyshev method
def chebyshev_method_general(u0_func, V_func, L, N, dt, t_steps):
```

```
    x_cheb = L * chebpts1(N)
    u = u0_func(x_cheb)
    V = V_func(x_cheb)

    D = np.zeros((N, N))
    for i in range(N):
        for j in range(N):
            if i != j:
                D[i, j] = (-1)**(i + j) / (x_cheb[i] - x_cheb[j])
        D[i, i] = -np.sum(D[i, :])
    D2 = D @ D / np.max(np.abs(D @ D))

    u_history = [u.copy()]
    for _ in range(t_steps):
        u = u + dt * 1j * (D2 @ u + V * u)
        u_history.append(u.copy())
    return np.array(u_history), x_cheb


L = 10
N = 256
t_max = 1.0
dt = 0.001
t_steps = int(t_max / dt)
t_plot_idx = int(1.0 / dt)


u0_func = lambda x: 1 / np.cosh(x)


epsilons_extended = [-1.0, 0.0, 0.5, 1.0, 2.0]
bs = [0.5, 1.0, 1.5]

for b in bs:
    plt.figure(figsize=(12, 6))
    for eps in epsilons_extended:
```

```
        V_func = lambda x: np.abs(x)**2 - eps * np.abs(x)**b
        u_hist, x_cheb = chebyshev_method_general(u0_func, V_func, L, N, dt,
        plt.plot(x_cheb, np.abs(u_hist[t_plot_idx])**2, label=fr"$\varepsilon
    plt.title(fr"Chebyshev Method at $t=1.0$ for $b={b}$")
    plt.xlabel("x")
    plt.ylabel(r"$|u(x,t)|^2$")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

Listing 8: Crank-Nicolson Method with modified potential

```
from scipy.sparse import diags, identity
from scipy.sparse.linalg import splu

# Crank-Nicolson method implementation
def crank_nicolson_method(u0, V, dx, dt, t_steps):
    N = len(u0)
    u = u0.copy().astype(np.complex128)
    u_history = [u.copy()]

    main_diag = -2 * np.ones(N) / dx**2
    off_diag = np.ones(N - 1) / dx**2
    lap = diags([off_diag, main_diag, off_diag], [-1, 0, 1], format='csc')
    H = lap + diags(V, 0, format='csc')

    I = identity(N, format='csc', dtype=np.complex128)
    B = I - 0.5j * dt * H
    C = I + 0.5j * dt * H
    solver = splu(B)

    for _ in range(t_steps):
        u = solver.solve(C @ u)
        u_history.append(u.copy())
    return np.array(u_history)
```

```
L = 10
N = 256
x = np.linspace(-L, L, N)
dx = x[1] - x[0]
u0 = 1 / np.cosh(x)
dt = 0.001
t_steps = int(1.0 / dt)

epsilons = [0.5, 1.0, 2.0,0,-1]
bs = [0.5, 1.0, 1.5]

for b in bs:
    plt.figure(figsize=(12, 6))
    for eps in epsilons:
        V = np.abs(x)**2 - eps * np.abs(x)**b
        u_hist = crank_nicolson_method(u0, V, dx, dt, t_steps)
        plt.plot(x, np.abs(u_hist[-1])**2, label=fr"$\varepsilon$={eps}, $b$=
    plt.title(fr" C r a n k Nicolson  Method at $t=1.0$ for $b={b}$")
    plt.xlabel("x")
    plt.ylabel(r"$|u(x,t)|^2$")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

Listing 9: Perturbation theory with modified potential

```
from scipy.integrate import quad
from scipy.special import eval_hermite, factorial
import numpy as np
import matplotlib.pyplot as plt


L = 10
x = np.linspace(-L, L, 256)
t_vals = [0.0, 0.25, 0.5, 1.0]
```

```
N_terms = 40


def u0_func(x):
    return 1 / np.cosh(x)


def phi_n(n, x):
    Hn = eval_hermite(n, x)
    norm = np.sqrt(2**n * factorial(n) * np.sqrt(np.pi))
    return Hn * np.exp(-x**2 / 2) / norm

def compute_cn(n):
    integrand = lambda x: u0_func(x) * phi_n(n, x)
    result, _ = quad(integrand, -L, L)
    return result


def compute_E1(n, b):
    integrand = lambda x: np.abs(phi_n(n, x))**2 * np.abs(x)**b
    result, _ = quad(integrand, -L, L)
    return -result

# Perturbation theory approximation of u(x, t)
def perturbation_solution(x, t, epsilon, b):
    u = np.zeros_like(x, dtype=complex)
    for n in range(N_terms):
        cn = compute_cn(n)
        phi_vals = phi_n(n, x)
        E0 = 2 * n + 1
        E1 = compute_E1(n, b)
        phase = np.exp(-1j * (E0 + epsilon * E1) * t)
        u += cn * phi_vals * phase
    return u
```

```
epsilons = [0.5, 1.0, 2.0, -1.0, 0]
bs = [0.5, 1.0, 1.5]

for b in bs:
    plt.figure(figsize=(12, 6))
    for eps in epsilons:
        u_final = perturbation_solution(x, t=1.0, epsilon=eps, b=b)
        plt.plot(x, np.abs(u_final)**2, label=fr"$\varepsilon$={eps}, $b$={b}
    plt.title(fr"Perturbation Approximation at $t=1.0$ for $b={b}$")
    plt.xlabel("x")
    plt.ylabel(r"$|u(x,t)|^2$")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```