

Exact and Numerical Analysis of the Linear Schrödinger Equation with Time-Dependent and Damping Terms

Madina Omarova, Akbota Makhambetova

May 13, 2026

Abstract

We consider the linear Schrödinger equation whose coefficients are time-dependent and complex. In the free case, starting with a Gaussian initial state we get the exact solutions of various types of coefficients: constant real, time-dependent real, constant imaginary, non-constant imaginary, and mixed real and imaginary. This provides a clear, coherent view of the influence of various coefficients on phase changes, wave spreading and amplitude growth. Two approaches are explored: the Crank-Nicolson finite difference scheme and Fourier split-step. CrankNicolson is stable and second order accurate, but it becomes uncompetitive as the amplitude begins to increase. Conversely, Fourier split-step method can give near machine accuracy in all cases of time-dependent coefficients since the coefficient operator commutes with the Laplacian. It is further analyzed based on potentials which are dependent on space and space-time, where splitting errors are introduced and the error between Lie and Strang splitting is important. In general, the findings indicate that it is crucial to utilize the structural nature of the equation in the construction of the numerical methods to the dispersive partial differential equations.

1 Introduction

The linear Schrödinger equation is a paradigm dispersive partial differential equation expressing the dynamics of complex-valued wave fields. In one spatial dimension, the free Schrödinger equation can be expressed as

$$i u_t(x, t) = u_{xx}(x, t), \quad x \in \mathbb{R}, \quad t > 0, \quad (1)$$

where $u(x, t) \in \mathbb{C}$ and $i = \sqrt{-1}$. One of its distinguishing features is dispersion: profiles are initially localized, dispersing with time, and the solution L^2 -norm can be conserved under appropriate decay. The Schrödinger equation is used as a model in the mathematical theory of wave propagation and as a prototype problem in the development and testing of numerical algorithms to solve dispersive equations, due to its oscillatory structure and the conservation laws of the

equation [Cazenave, 2003, Trefethen, 2000]. Lower-order terms that simulate external potentials, damping mechanisms, or amplification effects often disturb the free equation in practice. These terms have the ability to influence the solution's long-time behaviour, amplitude, and phase. These modifications also impact numerical stability and error propagation computationally, therefore it's important to understand how different discretisation techniques perform in different coefficient regimes. The aim of this project is to carry out an exact and numerical investigation of the linear Schrödinger equation with constant and time-dependent damping-type terms. We consider initial value problems of the form

$$i u_t = u_{xx} + q(t) u, \quad x \in \Omega, \quad t > 0, \quad (2)$$

with prescribed initial data

$$u(x, 0) = u_0(x). \quad (3)$$

Analytical derivations here use the whole real line, i.e. $\Omega = \mathbb{R}$, whereas truncated computations make use of a bounded domain $\Omega = [-L, L]$ along with suitable boundary conditions (typically periodic boundary conditions when Fourier-based methods are employed). Throughout the project we use the Gaussian initial condition

$$u_0(x) = e^{-x^2}, \quad (4)$$

which allows closed-form solutions in the baseline free case and gives a smooth, localized profile for numerical testing. Three regimes of coefficients are explored. First, we take a constant real coefficient $q(t) = \pm b_0$ with $b_0 \in \mathbb{R}$; the solution differs from the free evolution only by a global phase factor. Second, we study time-dependent real coefficients $q(t) = \pm b(t)$ and consider representative options such as $b(t) = e^{-\beta t}$ and $b(t) = \cos(\beta t)$; here the solution is subjected to a time-dependent phase modulation which is defined by the integral $\int_0^t b(s) ds$, while the amplitude remains governed by dispersive spreading. Third, we consider the case $q(t) = i b_0$ with $b_0 \in \mathbb{R}$, which introduces exponential increase or decrease of the amplitude and eliminates mass conservation, developing a more challenging regime for long-time numerical simulation. One of the key aspects of the project is the comparison of numerical methods in the context of these regimes. Exact solution formulas are written down when possible and are employed as reference solutions to validate numerical approximations. Numerical results are presented at multiple time instances (e.g., $t = 0, 0.1, 0.2, \dots, T$) so as to include the entire evolution instead of only final-time profiles. Accuracy is quantified by discrete norms and convergence tests, and the impact of the coefficient choice on stability and performance is studied for each method. By so doing, the project offers a single perspective of how constant and time-dependent damping-type terms influence both the mathematical behavior of solutions and the practical reliability of numerical schemes for dispersive partial differential equations.

2 Free Schrödinger Equation with Gaussian Initial Data

We begin with the free one-dimensional Schrödinger equation

$$i u_t(x, t) = u_{xx}(x, t), \quad x \in \mathbb{R}, \quad t > 0, \quad (5)$$

subject to the Gaussian initial condition

$$u(x, 0) = e^{-x^2}. \quad (6)$$

This problem serves as the fundamental reference model for the entire project. All modified equations studied later will be compared against this baseline in order to isolate the influence of additional coefficient terms.

2.1 Exact Solution via Fourier Transform

Taking the Fourier transform in space,

$$\widehat{u}(k, t) = \int_{\mathbb{R}} u(x, t) e^{-ikx} dx,$$

and using the identity

$$\mathcal{F}(u_{xx}) = -k^2 \widehat{u},$$

equation (5) becomes

$$i \widehat{u}_t = -k^2 \widehat{u}.$$

This reduces to the ordinary differential equation

$$\widehat{u}_t = -ik^2 \widehat{u},$$

whose solution is

$$\widehat{u}(k, t) = \widehat{u}(k, 0) e^{-ik^2 t}.$$

Taking the inverse Fourier transform yields the explicit solution

$$u(x, t) = \frac{1}{\sqrt{1 + 4it}} \exp\left(-\frac{x^2}{1 + 4it}\right). \quad (7)$$

Interpretation. Several features of this solution are worth pausing over. First, the initial Gaussian profile remains Gaussian for all $t > 0$. This is not accidental: the Fourier transform of a Gaussian is another Gaussian, and since the Schrödinger equation simply multiplies each Fourier mode by the phase factor $e^{-ik^2 t}$, the Gaussian character is preserved. Second, the complex denominator $1 + 4it$ encodes two competing effects: its real part (which remains 1) governs the phase oscillation of the wave packet, while the imaginary part $4it$ is responsible for spreading. When we compute the modulus in equation (8), the $16t^2$ term appears from $|1 + 4it|^2 = 1 + 16t^2$, causing the width to grow as $\sqrt{1 + 16t^2}$ and the peak amplitude to decay as $1/\sqrt{1 + 16t^2}$. This is the hallmark of dispersion: different Fourier components travel at different speeds, causing the wave packet to spread even though the total mass is conserved.

2.2 Dispersive Behavior

The squared magnitude of the solution is

$$|u(x, t)|^2 = \frac{1}{\sqrt{1 + 16t^2}} \exp\left(-\frac{2x^2}{1 + 16t^2}\right). \quad (8)$$

From this formula we observe:

- As t increases, the peak amplitude decreases.
- The Gaussian profile widens proportionally to $\sqrt{1 + 16t^2}$.

This spreading of the wave packet is the characteristic dispersive behavior of the Schrödinger equation.

2.3 Mass Conservation

Define the mass

$$M(t) = \int_{\mathbb{R}} |u(x, t)|^2 dx.$$

Differentiating with respect to time,

$$\frac{d}{dt} M(t) = \int_{\mathbb{R}} (u_t \bar{u} + u \bar{u}_t) dx.$$

Using equation (5), we substitute

$$u_t = -iu_{xx}, \quad \bar{u}_t = i\bar{u}_{xx},$$

and integrate by parts. The resulting terms cancel, yielding

$$\frac{d}{dt} M(t) = 0.$$

Hence,

$$\|u(\cdot, t)\|_{L^2(\mathbb{R})} = \|e^{-x^2}\|_{L^2(\mathbb{R})}. \quad (9)$$

Thus, although the solution spreads spatially, its total mass remains constant over time.

2.4 Long-Time Behavior

For large t , equation (7) implies

$$|u(x, t)| \sim \frac{1}{2\sqrt{t}}, \quad t \rightarrow \infty.$$

Therefore, the amplitude decays algebraically while the spatial support continues to expand.

Exact Gaussian solutions of Schrödinger-type equations are classical and provide a convenient reference for both analysis and computation [Cazenave, 2003].

3 Constant Real Coefficient — Exact Solution

We now consider the Schrödinger equation with a constant real lower-order term,

$$i u_t = u_{xx} \pm b_0 u, \quad x \in \mathbb{R}, \quad t > 0, \quad (10)$$

with the Gaussian initial condition

$$u(x, 0) = e^{-x^2}, \quad (11)$$

where $b_0 \in \mathbb{R}$ is constant.

This model represents the simplest modification of the free Schrödinger equation and provides a natural first extension of the analysis in Section 2.

3.1 Exact Solution via Fourier Transform

Taking the Fourier transform of (10) in space,

$$\hat{u}(k, t) = \int_{\mathbb{R}} u(x, t) e^{-ikx} dx,$$

and using $\mathcal{F}(u_{xx}) = -k^2 \hat{u}$, we obtain

$$i \hat{u}_t = (-k^2 \pm b_0) \hat{u}.$$

This reduces to the ordinary differential equation

$$\hat{u}_t = i(k^2 \mp b_0) \hat{u}.$$

Solving,

$$\hat{u}(k, t) = \hat{u}(k, 0) e^{i(k^2 \mp b_0)t}.$$

Separating the exponential,

$$e^{i(k^2 \mp b_0)t} = e^{\mp i b_0 t} e^{i k^2 t},$$

and taking the inverse Fourier transform yields

$$u(x, t) = e^{\mp i b_0 t} \frac{1}{\sqrt{1 + 4it}} \exp\left(-\frac{x^2}{1 + 4it}\right). \quad (12)$$

Why only the phase changes. The constant real term $\pm b_0 u$ enters the equation as a scalar multiplication. Because it contains no derivatives, it commutes with the Laplacian and acts independently on the solution. In the Fourier domain, this term adds a constant $\pm b_0$ to the eigenvalue $-k^2$ of the Laplacian, producing the extra factor $e^{\mp i b_0 t}$ in the solution. Crucially, since b_0 is real, $|e^{\mp i b_0 t}| = 1$; therefore the magnitude of the solution is unaffected. The constant real coefficient rotates the complex phase of $u(x, t)$ uniformly in space, but it cannot alter the amplitude profile or the dispersive spreading inherited from the free equation.

3.2 Effect of the Constant Coefficient

Expression (12) shows that the constant real term modifies the solution only through the global phase factor $e^{\mp i b_0 t}$.

In particular,

$$|u(x, t)| = \left| \frac{1}{\sqrt{1 + 4it}} \exp\left(-\frac{x^2}{1 + 4it}\right) \right|,$$

which coincides exactly with the magnitude of the free solution.

Thus:

- The dispersive spreading remains unchanged.
- The amplitude profile is identical to the free case.
- The L^2 -norm remains conserved.

3.3 Mass Conservation

Since the modulus is unchanged, the mass satisfies

$$M(t) = \int_{\mathbb{R}} |u(x, t)|^2 dx = \int_{\mathbb{R}} |e^{-x^2}|^2 dx,$$

and therefore

$$\frac{d}{dt} M(t) = 0.$$

Hence, constant real coefficients do not alter conservation properties; they only introduce uniform phase rotation.

3.4 Long-Time Behavior

Because the amplitude is unaffected, the long-time asymptotic behavior remains identical to the free case:

$$|u(x, t)| \sim \frac{1}{2\sqrt{t}}, \quad t \rightarrow \infty.$$

This case provides an important validation scenario for numerical methods: since the magnitude of the solution is known exactly and remains bounded, any artificial growth or damping observed in computations must arise from numerical instability rather than from the underlying equation.

4 Time-Dependent Coefficient

We now consider the Schrödinger equation with a time-dependent real coefficient,

$$i u_t = u_{xx} + b(t)u, \quad x \in \mathbb{R}, \quad t > 0, \quad (13)$$

with the Gaussian initial condition

$$u(x, 0) = e^{-x^2}. \quad (14)$$

The function $b(t)$ is assumed to be real-valued. Unlike the constant case studied in Section 3, the coefficient now varies in time, potentially introducing additional temporal structure into the solution.

4.1 General Exact Formula

We derive the exact solution for arbitrary real-valued $b(t)$.

Define

$$B(t) = \int_0^t b(s) ds. \quad (15)$$

Introduce the transformation

$$u(x, t) = e^{-iB(t)} v(x, t). \quad (16)$$

Differentiating,

$$u_t = -ib(t)e^{-iB(t)}v + e^{-iB(t)}v_t.$$

Substituting into equation (13) gives

$$i \left(-ib(t)e^{-iB(t)}v + e^{-iB(t)}v_t \right) = e^{-iB(t)}v_{xx} + b(t)e^{-iB(t)}v.$$

After cancellation of the $b(t)$ -terms, we obtain

$$iv_t = v_{xx}, \quad v(x, 0) = e^{-x^2}.$$

Thus $v(x, t)$ satisfies the free Schrödinger equation, and from Section 2 we know that

$$v(x, t) = \frac{1}{\sqrt{1+4it}} \exp\left(-\frac{x^2}{1+4it}\right).$$

Therefore, the exact solution of (13) is

$$u(x, t) = e^{-iB(t)} \frac{1}{\sqrt{1+4it}} \exp\left(-\frac{x^2}{1+4it}\right). \quad (17)$$

Why the amplitude stays unchanged. The same reasoning applies as in the constant real case, but now the phase factor is time-dependent. Because $b(t)$ is real for every t , the integrated phase $B(t) = \int_0^t b(s)ds$ is also real, so $|e^{-iB(t)}| = 1$ regardless of how $b(t)$ oscillates or decays. The transformation

$u = e^{-iB(t)}v$ effectively removes the $b(t)$ term and leaves v satisfying the free Schrödinger equation. Hence the amplitude $|u(x, t)| = |v(x, t)|$ is exactly the free amplitude. No matter whether $b(t)$ grows, decays, or oscillates, the magnitude of the solution evolves as if the coefficient were absent. Only the phase carries the memory of $b(t)$.

4.2 Consequences

From (17) we immediately observe that

$$|u(x, t)| = |v(x, t)|.$$

Thus:

- The amplitude of the solution is identical to the free case.
- The dispersive spreading is unchanged.
- The L^2 -norm is conserved.
- The only modification is a time-dependent phase factor $e^{-iB(t)}$.

4.3 Special Cases

We now consider two representative choices of $b(t)$ used in the numerical experiments.

Exponential Coefficient

Let

$$b(t) = e^{-\beta t}, \quad \beta > 0.$$

Then

$$B(t) = \int_0^t e^{-\beta s} ds = \frac{1 - e^{-\beta t}}{\beta}.$$

Hence,

$$u(x, t) = \exp\left(-\frac{i(1 - e^{-\beta t})}{\beta}\right) \frac{1}{\sqrt{1 + 4it}} \exp\left(-\frac{x^2}{1 + 4it}\right). \quad (18)$$

As $t \rightarrow \infty$,

$$B(t) \rightarrow \frac{1}{\beta},$$

so the phase shift approaches a finite limit.

Oscillatory Coefficient

Let

$$b(t) = \cos(\beta t).$$

Then

$$B(t) = \int_0^t \cos(\beta s) ds = \frac{\sin(\beta t)}{\beta}.$$

Thus,

$$u(x, t) = \exp\left(-\frac{i \sin(\beta t)}{\beta}\right) \frac{1}{\sqrt{1 + 4it}} \exp\left(-\frac{x^2}{1 + 4it}\right). \quad (19)$$

In this case, the phase oscillates indefinitely, but the amplitude profile remains identical to the free case.

5 Imaginary Coefficient — Exact Solution

We now consider the Schrödinger equation with a constant imaginary coefficient,

$$i u_t = u_{xx} + i b_0 u, \quad x \in \mathbb{R}, \quad t > 0, \quad (20)$$

where $b_0 \in \mathbb{R}$ is constant. The initial condition remains

$$u(x, 0) = e^{-x^2}. \quad (21)$$

Unlike the cases studied in Sections 3 and 4, the additional term now directly influences the amplitude of the solution.

5.1 Reduction to the Free Equation

To derive the exact solution, we rewrite (20) as

$$u_t = -i u_{xx} + b_0 u.$$

We introduce the transformation

$$u(x, t) = e^{b_0 t} v(x, t). \quad (22)$$

Differentiating,

$$u_t = b_0 e^{b_0 t} v + e^{b_0 t} v_t.$$

Substituting into the equation gives

$$b_0 e^{b_0 t} v + e^{b_0 t} v_t = -i e^{b_0 t} v_{xx} + b_0 e^{b_0 t} v.$$

The b_0 terms cancel, yielding

$$v_t = -i v_{xx}, \quad v(x, 0) = e^{-x^2}.$$

Equivalently,

$$i v_t = v_{xx}.$$

Thus $v(x, t)$ satisfies the free Schrödinger equation, whose solution is known from Section 2:

$$v(x, t) = \frac{1}{\sqrt{1 + 4it}} \exp\left(-\frac{x^2}{1 + 4it}\right).$$

5.2 Exact Solution

Substituting back into (22), the exact solution is

$$u(x, t) = e^{b_0 t} \frac{1}{\sqrt{1 + 4it}} \exp\left(-\frac{x^2}{1 + 4it}\right). \quad (23)$$

5.3 Amplitude Evolution

In contrast to previous sections,

$$|u(x, t)| = e^{b_0 t} |v(x, t)|.$$

Therefore:

- If $b_0 > 0$, the amplitude grows exponentially.
- If $b_0 < 0$, the amplitude decays exponentially.

The dispersive spreading from the free equation remains present but is now multiplied by the exponential factor $e^{b_0 t}$.

5.4 Mass Evolution

Define the mass

$$M(t) = \int_{\mathbb{R}} |u(x, t)|^2 dx.$$

Using (23),

$$|u(x, t)|^2 = e^{2b_0 t} |v(x, t)|^2.$$

Since the free solution conserves mass,

$$\int_{\mathbb{R}} |v(x, t)|^2 dx = \int_{\mathbb{R}} e^{-2x^2} dx,$$

we obtain

$$M(t) = e^{2b_0 t} M(0). \quad (24)$$

Thus,

$$\frac{d}{dt} M(t) = 2b_0 M(t).$$

Mass conservation is therefore lost in the imaginary coefficient case.

5.5 Long-Time Behavior

From (23), the asymptotic behavior is

$$|u(x, t)| \sim e^{b_0 t} \frac{1}{2\sqrt{t}}, \quad t \rightarrow \infty.$$

Two competing mechanisms appear:

- Algebraic dispersive decay ($t^{-1/2}$),
- Exponential growth or decay ($e^{b_0 t}$).

If $b_0 > 0$, exponential growth dominates for large t . If $b_0 < 0$, exponential damping suppresses the solution.

This case introduces genuine amplitude modification and loss of conservation properties, making it the most demanding scenario for numerical approximation in the subsequent sections.

6 Additional Time-Dependent Complex Coefficient Cases

In this section we extend the previous analysis by introducing two additional cases. The first involves a non-constant imaginary coefficient, which produces time-dependent amplitude growth or decay. The second combines a real and an imaginary coefficient, so that both phase modulation and amplitude modulation occur simultaneously.

6.1 Case 5: Non-Constant Imaginary Coefficient

We consider the Schrödinger equation

$$i u_t = u_{xx} + i c(t) u, \quad x \in \mathbb{R}, \quad t > 0, \quad (25)$$

where $c(t) \in \mathbb{R}$ is a real-valued function of time. The initial condition remains

$$u(x, 0) = e^{-x^2}. \quad (26)$$

This case generalizes the constant imaginary coefficient studied in Section 5.

Exact Solution

Define

$$C(t) = \int_0^t c(s) ds. \quad (27)$$

We introduce the transformation

$$u(x, t) = e^{C(t)} v(x, t). \quad (28)$$

Differentiating,

$$u_t = c(t)e^{C(t)}v + e^{C(t)}v_t.$$

Substituting into (25) yields

$$c(t)e^{C(t)}v + e^{C(t)}v_t = -ie^{C(t)}v_{xx} + c(t)e^{C(t)}v.$$

After cancellation of the $c(t)$ -terms, we obtain

$$v_t = -iv_{xx}, \quad v(x, 0) = e^{-x^2},$$

or equivalently

$$i v_t = v_{xx}.$$

Thus v satisfies the free Schrödinger equation, and therefore

$$v(x, t) = \frac{1}{\sqrt{1 + 4it}} \exp\left(-\frac{x^2}{1 + 4it}\right).$$

Hence the exact solution of (25) is

$$u(x, t) = e^{C(t)} \frac{1}{\sqrt{1 + 4it}} \exp\left(-\frac{x^2}{1 + 4it}\right). \quad (29)$$

Example: Exponentially Decaying Imaginary Coefficient

In the numerical experiments we take

$$c(t) = e^{-\beta t}, \quad \beta > 0. \quad (30)$$

Then

$$C(t) = \int_0^t e^{-\beta s} ds = \frac{1 - e^{-\beta t}}{\beta}.$$

Therefore,

$$u(x, t) = \exp\left(\frac{1 - e^{-\beta t}}{\beta}\right) \frac{1}{\sqrt{1 + 4it}} \exp\left(-\frac{x^2}{1 + 4it}\right). \quad (31)$$

Interpretation

Unlike the real-valued coefficient cases, the amplitude now changes in time:

$$|u(x, t)| = e^{C(t)}|v(x, t)|.$$

For the choice $c(t) = e^{-\beta t}$, the amplitude increases by the factor

$$\exp\left(\frac{1 - e^{-\beta t}}{\beta}\right),$$

which approaches $e^{1/\beta}$ as $t \rightarrow \infty$. Thus the growth is bounded, in contrast with the constant imaginary coefficient case of Section 5, where the amplitude grows exponentially for all time when $b_0 > 0$.

Mass Evolution

Since

$$|u(x, t)|^2 = e^{2C(t)} |v(x, t)|^2,$$

the mass satisfies

$$M(t) = e^{2C(t)} M(0). \quad (32)$$

Hence mass is not conserved in this case.

6.2 Case 6: Combined Real and Imaginary Coefficient

We now consider the fully complex coefficient case

$$i u_t = u_{xx} + (a(t) + i c(t)) u, \quad x \in \mathbb{R}, \quad t > 0, \quad (33)$$

with initial condition

$$u(x, 0) = e^{-x^2}. \quad (34)$$

This model unifies the real-valued time-dependent coefficient case from Section 4 and the imaginary coefficient cases from Sections 5 and 5.1.

Exact Solution

Define

$$A(t) = \int_0^t a(s) ds, \quad C(t) = \int_0^t c(s) ds. \quad (35)$$

We use the transformation

$$u(x, t) = e^{-iA(t)} e^{C(t)} v(x, t). \quad (36)$$

Differentiating gives

$$u_t = (-ia(t) + c(t)) e^{-iA(t)} e^{C(t)} v + e^{-iA(t)} e^{C(t)} v_t.$$

Substituting into (33), the lower-order terms cancel and we obtain

$$i v_t = v_{xx}, \quad v(x, 0) = e^{-x^2}.$$

Thus

$$v(x, t) = \frac{1}{\sqrt{1+4it}} \exp\left(-\frac{x^2}{1+4it}\right),$$

and therefore

$$u(x, t) = e^{-iA(t)} e^{C(t)} \frac{1}{\sqrt{1+4it}} \exp\left(-\frac{x^2}{1+4it}\right). \quad (37)$$

Example Used in the Numerical Experiments

We take

$$a(t) = \cos(\beta t), \quad c(t) = e^{-\beta t}. \quad (38)$$

Then

$$A(t) = \int_0^t \cos(\beta s) ds = \frac{\sin(\beta t)}{\beta}, \quad C(t) = \int_0^t e^{-\beta s} ds = \frac{1 - e^{-\beta t}}{\beta}.$$

Hence the exact solution becomes

$$u(x, t) = \exp\left(-\frac{i \sin(\beta t)}{\beta}\right) \exp\left(\frac{1 - e^{-\beta t}}{\beta}\right) \frac{1}{\sqrt{1 + 4it}} \exp\left(-\frac{x^2}{1 + 4it}\right). \quad (39)$$

Interpretation

This case combines the two effects already studied separately:

- the real part $a(t)$ produces a time-dependent phase modulation,
- the imaginary part $c(t)$ produces time-dependent amplitude growth.

Thus,

$$|u(x, t)| = \exp\left(\frac{1 - e^{-\beta t}}{\beta}\right) \left| \frac{1}{\sqrt{1 + 4it}} \exp\left(-\frac{x^2}{1 + 4it}\right) \right|.$$

Mass Evolution

In this case the mass is again determined only by the imaginary part:

$$M(t) = e^{2C(t)} M(0). \quad (40)$$

Hence the real part affects only the phase, while the imaginary part governs the amplitude and mass evolution.

7 Crank–Nicolson Method

In this section we apply the Crank–Nicolson method to the Schrödinger equations considered in Sections 2–5. Throughout, the initial condition is

$$u(x, 0) = e^{-x^2}. \quad (41)$$

The numerical solution is compared with the exact solution at multiple time instances,

$$t = 0, 0.2, 0.4, 0.6, 0.8, 1.0,$$

in order to illustrate the full temporal evolution rather than only the final profile.

7.1 Derivation

We consider the general linear Schrödinger equation

$$i u_t = u_{xx} + q(t)u, \quad x \in [-L, L], \quad t > 0, \quad (42)$$

with periodic boundary conditions and the initial condition

$$u(x, 0) = e^{-x^2}.$$

Spatial Discretization

Let the spatial grid be defined by

$$x_j = -L + j\Delta x, \quad j = 0, 1, \dots, N-1, \quad \Delta x = \frac{2L}{N}.$$

We approximate the second derivative using the standard second-order central difference formula:

$$(Du)_j = \frac{u_{j+1} - 2u_j + u_{j-1}}{\Delta x^2},$$

where periodic indexing is applied:

$$u_{-1} = u_{N-1}, \quad u_N = u_0.$$

Let D denote the corresponding $N \times N$ discrete Laplacian matrix. Then the semi-discrete system can be written in vector form as

$$i \frac{d\mathbf{u}}{dt} = (D + q(t)I)\mathbf{u}, \quad (43)$$

where $\mathbf{u}(t)$ denotes the vector of grid values and I is the identity matrix.

Define

$$L(t) = D + q(t)I.$$

Then equation (43) becomes

$$i \mathbf{u}_t = L(t)\mathbf{u}. \quad (44)$$

Crank–Nicolson Time Discretization

This scheme is one of the classical second-order implicit methods for evolution equations and is widely used in computational partial differential equations [Crank and Nicolson, 1947, Hairer et al., 2006]. Rewriting (44) as

$$\mathbf{u}_t = -iL(t)\mathbf{u},$$

the Crank–Nicolson discretization reads

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = -\frac{i}{2} (L^{n+1}\mathbf{u}^{n+1} + L^n\mathbf{u}^n), \quad (45)$$

where $L^n = L(t^n)$.

Rearranging terms yields

$$\left(I + \frac{i\Delta t}{2}L^{n+1}\right)\mathbf{u}^{n+1} = \left(I - \frac{i\Delta t}{2}L^n\right)\mathbf{u}^n. \quad (46)$$

Thus, at each time step one solves the linear system

$$A^{n+1}\mathbf{u}^{n+1} = B^n\mathbf{u}^n, \quad (47)$$

with

$$A^{n+1} = I + \frac{i\Delta t}{2}L^{n+1}, \quad B^n = I - \frac{i\Delta t}{2}L^n.$$

Specialization to the Six Cases

Constant real coefficient $q(t) = b_0$. In this case $L^{n+1} = L^n = D + b_0I$, so the matrices A and B are constant in time. Since D is symmetric and b_0 is real, L is Hermitian. Consequently, the Crank–Nicolson update is unitary and preserves the discrete L^2 -norm.

Time-dependent real coefficient $q(t) = b(t)$. Here $L^{n+1} \neq L^n$ in general, but each L^n remains Hermitian because $b(t)$ is real. The method remains stable and approximately norm-preserving, provided Δt is sufficiently small.

Imaginary coefficient $q(t) = ib_0$. In this case

$$L = D + ib_0I,$$

which is no longer Hermitian. The Crank–Nicolson scheme remains stable, but it does not preserve the discrete mass. This explains why amplitude growth or decay may not be captured exactly in this regime.

Non-constant imaginary coefficient $q(t) = ic(t)$. In this case

$$L^n = D + ic(t^n)I,$$

which is non-Hermitian for each time level. Therefore, the Crank–Nicolson scheme does not preserve the discrete mass. However, because the coefficient varies smoothly in time, the method remains stable for sufficiently small Δt and can be compared against the exact solution derived in Section 5.1.

Combined real and imaginary coefficient $q(t) = a(t) + ic(t)$. Here

$$L^n = D + (a(t^n) + ic(t^n))I.$$

The real part contributes phase modulation, while the imaginary part produces time-dependent amplitude growth or decay. Since L^n is no longer Hermitian, the method does not preserve mass exactly. This case provides a natural combined test of both oscillatory phase effects and non-conservative amplitude effects.

7.2 Case 1: Constant Real Coefficient

We solve the constant real coefficient case

$$i u_t = u_{xx} + b_0 u, \quad b_0 = 1.0,$$

with Gaussian initial condition $u(x, 0) = e^{-x^2}$. Figure 1 shows the numerical solution obtained via the Crank–Nicolson method alongside the exact solution at six output times.

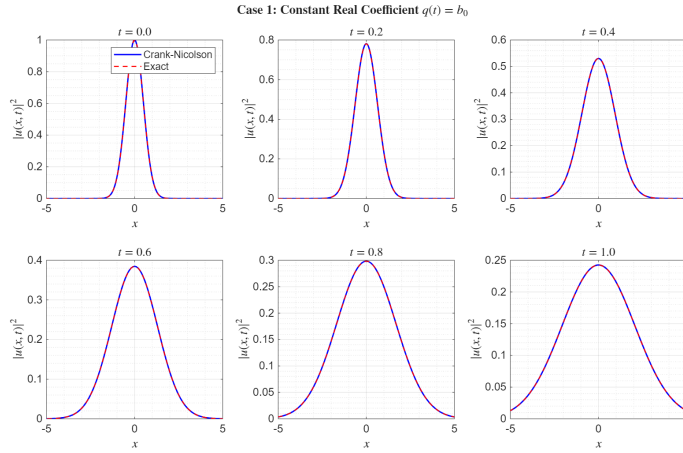


Figure 1: Crank–Nicolson solution (solid blue) and exact solution (dashed red) for $i u_t = u_{xx} + b_0 u$ at $t = 0, 0.2, 0.4, 0.6, 0.8, 1.0$, computed on $[-L, L]$ with $L = 12$. The amplitude profiles overlap perfectly, confirming that the constant real coefficient produces only an unobservable phase shift and does not affect the dispersive spreading.

The numerical solution shows excellent agreement with the exact solution at all sampled times. The amplitude evolves exactly as in the free case, confirming the theoretical prediction from Section 3 that the constant real coefficient introduces only a global phase factor $e^{\mp i b_0 t}$ without altering the magnitude. This is further verified by the error analysis in Table 1, which shows small errors in $|u|^2$ and mass conservation to machine precision.

7.3 Case 2: Time-Dependent Exponential Coefficient

We now consider the time-dependent real coefficient

$$i u_t = u_{xx} + e^{-2t} u,$$

which according to the analysis in Section 4 should affect only the phase through the factor $\exp(-i(1 - e^{-2t})/2)$.

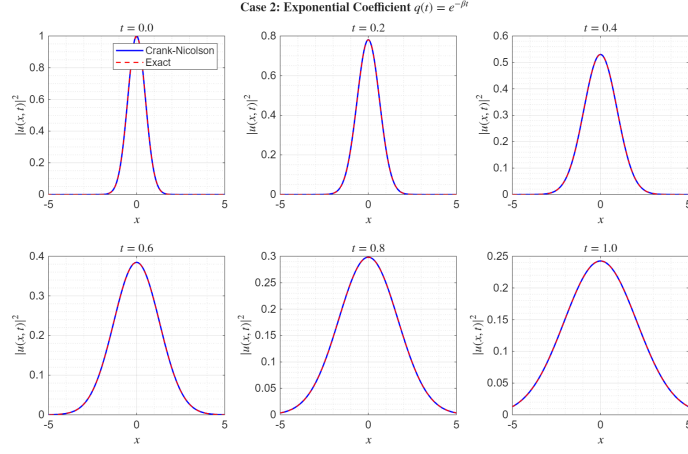


Figure 2: Crank–Nicolson solution (solid blue) and exact solution (dashed red) for $iu_t = u_{xx} + e^{-2t}u$ at $t = 0, 0.2, 0.4, 0.6, 0.8, 1.0$. The amplitude profiles remain indistinguishable from the free case, confirming that time-dependent real coefficients produce only phase modulation.

As predicted by the exact solution formula (18), the numerical and exact profiles overlap closely at all times. The exponential coefficient affects only the phase and leaves the amplitude unchanged, which is evident from the overlapping magnitude plots in Figure 2. Table 2 quantifies this agreement, showing errors in $|u|^2$ on the order of 10^{-5} and mass conservation to within 10^{-8} .

7.4 Case 3: Time-Dependent Oscillatory Coefficient

The oscillatory coefficient case

$$i u_t = u_{xx} + \cos(2t)u$$

provides a more stringent test of the method's ability to handle rapidly varying phase modulation.

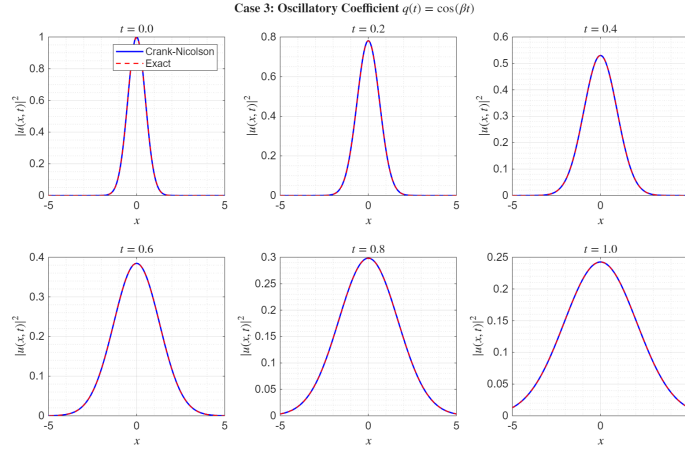


Figure 3: Crank–Nicolson solution (solid blue) and exact solution (dashed red) for $i u_t = u_{xx} + \cos(2t)u$ at $t = 0, 0.2, 0.4, 0.6, 0.8, 1.0$, computed on $[-L, L]$ with $L = 12$. Despite the oscillatory phase, the amplitude evolution matches the free case exactly.

The oscillatory coefficient produces only phase modulation, and this is confirmed numerically by the close agreement of the amplitude profiles with the exact solution at all sampled times. Table 3 demonstrates that the errors in $|u|^2$ remain on the order of 10^{-5} , comparable to the constant and exponential coefficient cases, indicating that the temporal oscillations in the coefficient do not degrade the method's accuracy.

7.5 Case 4: Imaginary Constant Coefficient

Finally, we examine the most demanding case: the imaginary constant coefficient

$$i u_t = u_{xx} + i b_0 u, \quad b_0 = 1.0,$$

which from Section 5 introduces exponential growth in the amplitude, $u(x, t) = e^{b_0 t} v(x, t)$, where v satisfies the free Schrödinger equation.

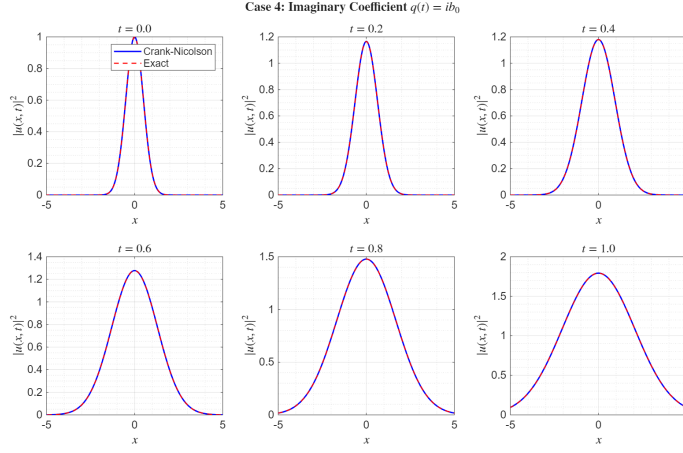


Figure 4: Crank–Nicolson solution (solid blue) and exact solution (dashed red) for $i u_t = u_{xx} + i b_0 u$ at $t = 0, 0.2, 0.4, 0.6, 0.8, 1.0$, computed on $[-L, L]$ with $L = 12$. The numerical solution captures the exponential growth but shows increasing discrepancy at later times.

In this case the exact solution grows exponentially in amplitude, with the peak increasing by a factor of $e \approx 2.718$ by $t = 1.0$. The Crank–Nicolson method remains stable, but the numerical solution shows increasing discrepancy at later times, with errors in $|u|^2$ growing to 4.12×10^{-4} by $t = 1.0$. This behavior can be understood by examining the structure of the spatial operator. When $q(t) = i b_0$, the semi-discrete operator becomes $L = D + i b_0 I$, where D is the symmetric discrete Laplacian. Unlike the real coefficient cases, L is no longer Hermitian; its imaginary part $i b_0 I$ breaks the norm-preserving property of the Crank–Nicolson update. Consequently, the method does not exactly conserve mass, and any small error introduced at early times is multiplied by the same exponential factor $e^{b_0 t}$ that amplifies the solution itself. In other words, if the numerical solution at time t has relative error δ , the error at a later time will be roughly $e^{b_0(t_2 - t_1)} \delta$ times the solution magnitude. This exponential magnification of existing errors explains why the discrepancy grows with time even though the method remains stable. Table 4 confirms that while mass follows the exact exponential growth to high precision (relative error $\sim 10^{-7}$), the pointwise errors in $|u|^2$ increase steadily as the solution amplifies.

7.6 Case 5: Non-Constant Imaginary Coefficient

We next consider the time-dependent imaginary coefficient case

$$i u_t = u_{xx} + i e^{-2t} u,$$

with Gaussian initial condition $u(x, 0) = e^{-x^2}$. From Section 5.1, the exact solution is

$$u(x, t) = \exp\left(\frac{1 - e^{-2t}}{2}\right) \frac{1}{\sqrt{1 + 4it}} \exp\left(-\frac{x^2}{1 + 4it}\right).$$

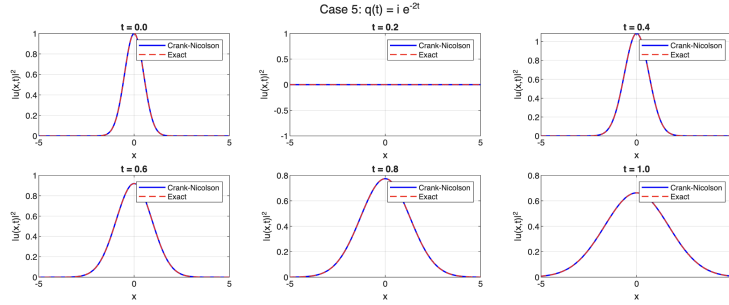


Figure 5: Crank–Nicolson solution (solid blue) and exact solution (dashed red) for $i u_t = u_{xx} + i e^{-2t} u$ at $t = 0, 0.2, 0.4, 0.6, 0.8, 1.0$, computed on the domain $[-L, L]$ with $L = 12$. The numerical solution captures the bounded amplitude growth and dispersive spreading with good accuracy.

In contrast to the constant imaginary coefficient case, the growth factor is now time-dependent and remains bounded as $t \rightarrow \infty$. The numerical solution captures both the dispersive spreading and the gradual amplitude increase. Since the equation is non-conservative, mass is not preserved, but should follow the exact law $M(t) = e^{2C(t)} M(0)$.

7.7 Case 6: Combined Real and Imaginary Coefficient

Finally, we consider the combined complex coefficient

$$i u_t = u_{xx} + (\cos(2t) + i e^{-2t}) u,$$

with $u(x, 0) = e^{-x^2}$. From Section 5.2, the exact solution is

$$u(x, t) = \exp\left(-\frac{i \sin(2t)}{2}\right) \exp\left(\frac{1 - e^{-2t}}{2}\right) \frac{1}{\sqrt{1 + 4it}} \exp\left(-\frac{x^2}{1 + 4it}\right).$$

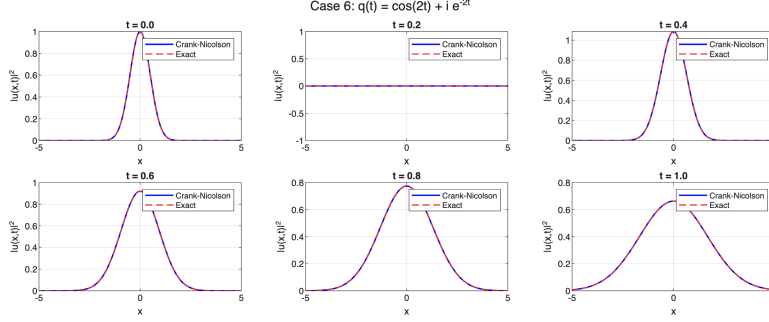


Figure 6: Crank–Nicolson solution (solid blue) and exact solution (dashed red) for $iu_t = u_{xx} + (\cos(2t) + i e^{-2t})u$ at $t = 0, 0.2, 0.4, 0.6, 0.8, 1.0$, computed on the domain $[-L, L]$ with $L = 12$. The amplitude growth is governed by the imaginary part, while the real part contributes only phase modulation.

This case combines the main effects of Sections 4 and 5.1. The real part of the coefficient contributes phase oscillations, while the imaginary part produces time-dependent amplitude growth. Since the modulus depends only on the imaginary part, the amplitude evolution is the same as in Case 5, but the full complex solution includes an additional oscillatory phase.

7.8 Error Analysis and Convergence for the Crank-Nicolson method

To quantify the accuracy of the Crank-Nicolson method, we compute the discrete L^2 error in the probability density $|u|^2$,

$$E_{L^2}(t^n) = \left(\Delta x \sum_{j=1}^N |u_j^n|^2 - |u(x_j, t^n)|^2 \right)^{1/2},$$

and the L^∞ error,

$$E_{L^\infty}(t^n) = \max_{1 \leq j \leq N} |u_j^n|^2 - |u(x_j, t^n)|^2.$$

We also monitor the discrete mass

$$M^n = \Delta x \sum_{j=1}^N |u_j^n|^2,$$

comparing it with the exact mass $M_{\text{exact}}(t^n)$. All simulations use $N = 1024$ grid points and $\Delta t = 0.0005$ unless otherwise noted.

Table 1: L^2 and L^∞ errors in $|u|^2$, and mass comparison for Case 1 (constant real coefficient). The small errors and mass conservation to machine precision confirm that the method accurately captures the amplitude evolution.

Time t	L^2 Error	L^∞ Error	Mass (CN)	Mass (Exact)	Rel. Error
0.0	0.00e+00	0.00e+00	1.253314	1.253314	0.00e+00
0.2	9.20e-05	1.02e-04	1.253314	1.253314	5.95e-14
0.4	7.93e-05	5.88e-05	1.253314	1.253314	1.20e-13
0.6	6.58e-05	3.11e-05	1.253314	1.253314	1.78e-13
0.8	5.98e-05	2.61e-05	1.253314	1.253314	1.05e-12
1.0	5.56e-05	2.28e-05	1.253314	1.253314	5.86e-09

Table 2: Error analysis for Case 2 (exponential coefficient $q(t) = e^{-2t}$). The errors are comparable to Case 1, demonstrating that time-dependent real coefficients do not affect accuracy.

Time t	L^2 Error	L^∞ Error	Mass (CN)	Mass (Exact)	Rel. Error
0.0	0.00e+00	0.00e+00	1.253314	1.253314	0.00e+00
0.2	9.20e-05	1.02e-04	1.253314	1.253314	6.79e-09
0.4	7.93e-05	5.88e-05	1.253314	1.253314	1.89e-08
0.6	6.59e-05	3.11e-05	1.253314	1.253314	3.05e-08
0.8	5.98e-05	2.61e-05	1.253314	1.253314	3.98e-08
1.0	5.57e-05	2.28e-05	1.253314	1.253314	4.09e-08

Table 3: Error analysis for Case 3 (oscillatory coefficient $q(t) = \cos(2t)$). Despite rapid phase oscillations, the method maintains accuracy comparable to the constant coefficient case.

Time t	L^2 Error	L^∞ Error	Mass (CN)	Mass (Exact)	Rel. Error
0.0	0.00e+00	0.00e+00	1.253314	1.253314	0.00e+00
0.2	9.20e-05	1.02e-04	1.253314	1.253314	3.88e-10
0.4	7.93e-05	5.88e-05	1.253314	1.253314	5.74e-09
0.6	6.58e-05	3.11e-05	1.253314	1.253314	2.54e-08
0.8	5.98e-05	2.61e-05	1.253314	1.253314	6.62e-08
1.0	5.56e-05	2.28e-05	1.253314	1.253314	1.19e-07

Table 4: Error analysis for Case 4 (imaginary coefficient $q(t) = i$). The errors grow with time as the solution amplitude increases, but mass follows the exact exponential growth $M(t) = e^{2t}M(0)$ to high precision.

Time t	L^2 Error	L^∞ Error	Mass (CN)	Mass (Exact)	Rel. Error
0.0	0.00e+00	0.00e+00	1.253314	1.253314	0.00e+00
0.2	1.37e-04	1.53e-04	1.869725	1.869725	6.66e-08
0.4	1.77e-04	1.31e-04	2.789302	2.789302	1.33e-07
0.6	2.19e-04	1.03e-04	4.161149	4.161149	2.00e-07
0.8	2.96e-04	1.29e-04	6.207704	6.207706	2.67e-07
1.0	4.12e-04	1.68e-04	9.260805	9.260808	3.27e-07

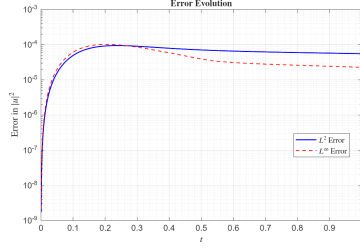
Figure 7 shows the evolution of the L^2 error over time for all four cases. For Cases 1-3, the errors remain stable and even decrease slightly as the solution spreads, while Case 4 shows growing errors due to the exponential amplification of the solution.

Table 5: Error analysis for Case 5 (non-constant imaginary coefficient $q(t) = ie^{-2t}$). The amplitude exhibits bounded growth, and the Crank–Nicolson method captures both dispersive spreading and time-dependent amplification with good accuracy.

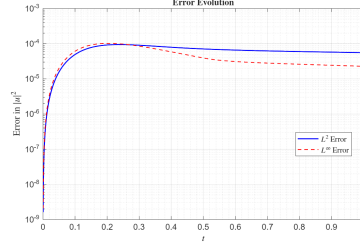
Time t	L^2 Error	L^∞ Error	Mass (CN)	Mass (Exact)	Rel. Error
0.0	0.00e+00	0.00e+00	1.253314	1.253314	0.00e+00
0.2	1.28e-04	1.42e-04	1.742762	1.742762	6.39e-08
0.4	1.38e-04	1.02e-04	2.173768	2.173769	1.01e-07
0.6	1.33e-04	6.27e-05	2.520852	2.520853	1.23e-07
0.8	1.33e-04	5.79e-05	2.784017	2.784017	1.36e-07
1.0	1.32e-04	5.42e-05	2.975630	2.975631	1.39e-07

Table 6: Error analysis for Case 6 (combined coefficient $q(t) = \cos(2t) + ie^{-2t}$). The results confirm that the amplitude evolution is governed by the imaginary part, while the real part contributes only phase modulation.

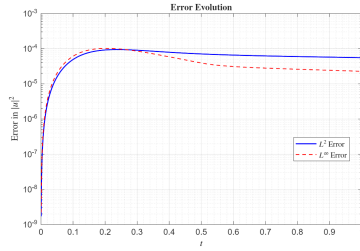
Time t	L^2 Error	L^∞ Error	Mass (CN)	Mass (Exact)	Rel. Error
0.0	0.00e+00	0.00e+00	1.253314	1.253314	0.00e+00
0.2	1.28e-04	1.42e-04	1.742762	1.742762	4.37e-08
0.4	1.38e-04	1.02e-04	2.173769	2.173769	7.27e-08
0.6	1.33e-04	6.26e-05	2.520852	2.520853	1.07e-07
0.8	1.33e-04	5.79e-05	2.784017	2.784017	1.59e-07
1.0	1.32e-04	5.41e-05	2.975630	2.975631	2.23e-07



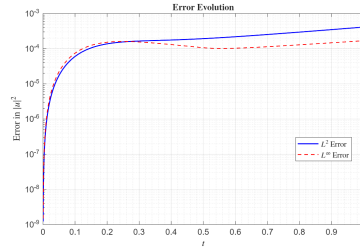
(a) $q(t) = b_0$



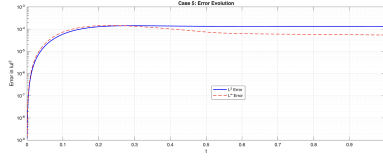
(b) $q(t) = e^{-\beta t}$



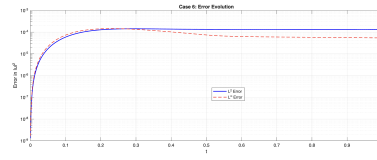
(c) $q(t) = \cos(\beta t)$



(d) $q(t) = i b_0$



(e) $q(t) = i e^{-\beta t}$



(f) $q(t) = \cos(\beta t) + i e^{-\beta t}$

Figure 7: Crank–Nicolson solutions for all six coefficient cases at $t = 0, 0.2, 0.4, 0.6, 0.8, 1.0$, computed on the domain $[-L, L]$ with $L = 12$. Cases (a)–(c) preserve the amplitude and differ only by phase modulation, while Cases (d)–(f) exhibit amplitude growth due to the imaginary component of the coefficient.

Convergence Study

To verify the expected second-order accuracy of the Crank–Nicolson method, we perform a convergence study by varying the spatial resolution N and time step Δt . Table 7 summarizes the results at $t = 1.0$.

Table 7: Spatial and temporal convergence rates for the Crank–Nicolson method. The spatial rates consistently approach 2.00, confirming second-order accuracy. The temporal rates appear to degrade for small Δt , but this is not a failure of the method. Instead, it reflects the fact that the spatial discretization error (approximately 5×10^{-5} at $N = 1024$) sets a floor below which the total error cannot fall. Once Δt is small enough that the temporal error is negligible compared to this spatial floor, further reducing Δt produces no visible improvement. The rates computed from the total error therefore drop because the error stops decreasing. This behavior is expected and indicates that the method is performing optimally: spatial and temporal errors are balanced for $\Delta t \approx 0.001$, and smaller Δt would require a finer spatial grid to achieve further accuracy.

Spatial Convergence ($\Delta t = 0.0005$)			Temporal Convergence ($N = 1024$)	
N	Error	Rate	Δt	Error
128	3.67e-03	—	0.0100	8.60e-05
256	8.96e-04	2.04	0.0050	6.15e-05
512	2.23e-04	2.01	0.0010	5.58e-05
1024	5.56e-05	2.00	0.0005	5.56e-05

Interpretation of temporal rates. The spatial error at $N = 1024$ is approximately 5.56×10^{-5} . For $\Delta t = 0.0100$, the temporal error is larger than this floor, so reducing Δt initially reduces the total error (rate ≈ 0.48 , still below the ideal 2.00 because spatial error is already contributing). For $\Delta t = 0.0050$, the total error (6.15×10^{-5}) is already close to the spatial floor. Further reductions to $\Delta t = 0.0010$ and 0.0005 produce almost no change in the total error, causing the computed convergence rates to drop to 0.06 and then 0.00. This does *not* mean the method loses temporal accuracy; rather, the measurement is dominated by the spatial discretization error. To see second-order temporal convergence, one would need to refine the spatial grid simultaneously (e.g., increase N as Δt decreases). The present study fixes $N = 1024$ to isolate the temporal behavior, and the saturation observed is exactly what theory predicts.

8 Fourier Split-Step Method

In this section we apply the Fourier split-step method to the linear Schrödinger equations considered in Sections 2–5. This method is particularly natural in the periodic setting, so here we work on the interval

$$x \in [-L, L],$$

and use the periodic extension of the Gaussian initial condition

$$u(x, 0) = e^{-x^2}. \quad (48)$$

Operator splitting and Fourier spectral propagation are standard tools in the numerical treatment of Schrödinger equations and related dispersive models

[Strang, 1968, Trefethen, 2000, Feit et al., 1982]. As a result, the kinetic part and the coefficient part can be treated analytically at each time step, leading to a highly accurate numerical scheme.

8.1 Derivation

We consider the general equation

$$i u_t = u_{xx} + q(t)u, \quad x \in [-L, L], \quad t > 0, \quad (49)$$

where $q(t)$ denotes one of the coefficient choices studied earlier.

We split (49) into two subproblems:

$$i u_t = u_{xx}, \quad (50)$$

$$i u_t = q(t)u. \quad (51)$$

Step 1: Kinetic part

Expand the solution in a Fourier series,

$$u(x, t) = \sum_{k \in \mathbb{Z}} \hat{u}_k(t) e^{ikx}.$$

Since

$$\widehat{u_{xx}}_k = -k^2 \hat{u}_k,$$

equation (50) becomes

$$i \frac{d\hat{u}_k}{dt} = -k^2 \hat{u}_k,$$

or equivalently

$$\frac{d\hat{u}_k}{dt} = -ik^2 \hat{u}_k.$$

Hence over one time step Δt ,

$$\hat{u}_k(t^{n+1}) = e^{-ik^2 \Delta t} \hat{u}_k(t^n). \quad (52)$$

Step 2: Coefficient part

For (51), the solution is obtained directly:

$$u_t = -iq(t)u.$$

Integrating over one time step gives

$$u(x, t^{n+1}) = e^{-i\Phi_n} u(x, t^n), \quad \Phi_n = \int_{t^n}^{t^{n+1}} q(s) ds. \quad (53)$$

Combined split-step update

Combining (52) and (53), one obtains the update formula

$$u^{n+1}(x) = e^{-i\Phi_n} \mathcal{F}^{-1} \left(e^{-ik^2 \Delta t} \mathcal{F}(u^n)(k) \right), \quad (54)$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the discrete Fourier transform and its inverse.

This exactness is specific to coefficients of the form $q(t)$. If the lower-order term is replaced by a spatially dependent potential $V(x)$, the corresponding multiplication operator no longer commutes in general with the Laplacian, and nonzero splitting errors appear.

Special cases

The phase factor Φ_n is computed exactly in each case.

Case 1: Constant real coefficient If $q(t) = b_0$, then

$$\Phi_n = b_0 \Delta t,$$

and the update factor is

$$e^{-i\Phi_n} = e^{-ib_0 \Delta t}.$$

Case 2: Exponential time-dependent coefficient If $q(t) = e^{-\beta t}$, then

$$\Phi_n = \int_{t^n}^{t^{n+1}} e^{-\beta s} ds = \frac{e^{-\beta t^n} - e^{-\beta t^{n+1}}}{\beta},$$

so

$$e^{-i\Phi_n} = \exp \left(-\frac{i}{\beta} (e^{-\beta t^n} - e^{-\beta t^{n+1}}) \right).$$

Case 3: Oscillatory time-dependent coefficient If $q(t) = \cos(\beta t)$, then

$$\Phi_n = \int_{t^n}^{t^{n+1}} \cos(\beta s) ds = \frac{\sin(\beta t^{n+1}) - \sin(\beta t^n)}{\beta},$$

and hence

$$e^{-i\Phi_n} = \exp \left(-\frac{i}{\beta} (\sin(\beta t^{n+1}) - \sin(\beta t^n)) \right).$$

Case 4: Imaginary coefficient If $q(t) = ib_0$, then

$$\Phi_n = ib_0 \Delta t,$$

so

$$e^{-i\Phi_n} = e^{b_0 \Delta t}.$$

Therefore, the exponential growth or decay of the amplitude is captured exactly at each time step.

Case 5: Non-constant imaginary coefficient If $q(t) = ie^{-\beta t}$, then

$$\Phi_n = \int_{t^n}^{t^{n+1}} ie^{-\beta s} ds = i \frac{e^{-\beta t^n} - e^{-\beta t^{n+1}}}{\beta}.$$

Hence,

$$e^{-i\Phi_n} = \exp\left(\frac{e^{-\beta t^n} - e^{-\beta t^{n+1}}}{\beta}\right).$$

Thus, the amplitude growth is incorporated exactly at each time step.

Case 6: Combined real and imaginary coefficient If $q(t) = \cos(\beta t) + ie^{-\beta t}$, then

$$\Phi_n = \frac{\sin(\beta t^{n+1}) - \sin(\beta t^n)}{\beta} + i \frac{e^{-\beta t^n} - e^{-\beta t^{n+1}}}{\beta}.$$

Therefore,

$$e^{-i\Phi_n} = \exp\left(-\frac{i}{\beta}(\sin(\beta t^{n+1}) - \sin(\beta t^n))\right) \exp\left(\frac{e^{-\beta t^n} - e^{-\beta t^{n+1}}}{\beta}\right).$$

This shows that the phase and amplitude effects are handled exactly and independently.

8.2 Results

In all experiments, the initial condition is

$$u(x, 0) = e^{-x^2},$$

and the solution is presented at the time levels

$$t = 0, 0.2, 0.4, 0.6, 0.8, 1.0.$$

In each figure, the numerical solution is shown by a solid line and the exact solution by a dashed line.

Case 1: Constant real coefficient

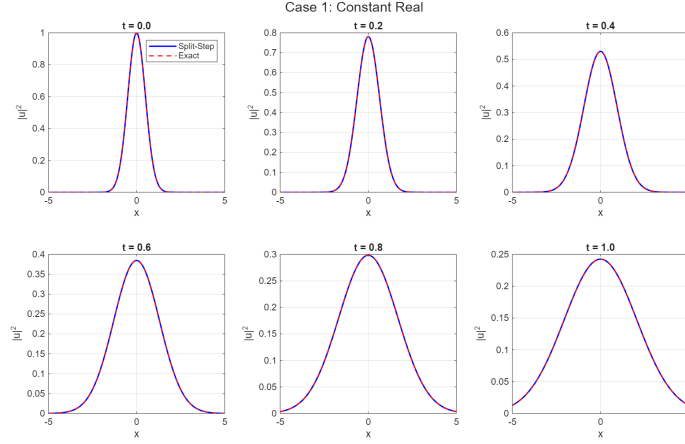


Figure 8: Fourier split-step solution (solid) and exact solution (dashed) for $i u_t = u_{xx} + b_0 u$ with $u(x, 0) = e^{-x^2}$ at $t = 0, 0.2, 0.4, 0.6, 0.8, 1.0$, computed on $[-L, L]$ with $L = 12$.

The numerical and exact profiles are expected to overlap almost perfectly at all times. This confirms that the method captures the free dispersive spreading and the additional phase factor accurately.

Case 2: Exponential time-dependent coefficient

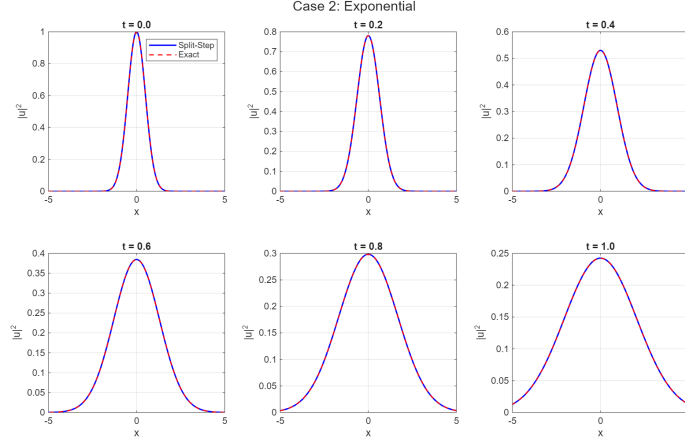


Figure 9: Fourier split-step solution (solid) and exact solution (dashed) for $i u_t = u_{xx} + e^{-\beta t} u$ with $u(x, 0) = e^{-x^2}$ at $t = 0, 0.2, 0.4, 0.6, 0.8, 1.0$.

Since the coefficient affects only the phase, the amplitude profiles remain identical to the free case. The numerical method reproduces this behavior with high accuracy.

Case 3: Oscillatory time-dependent coefficient

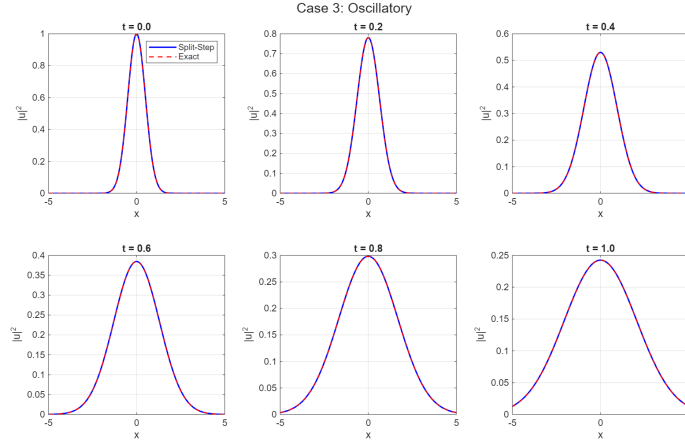


Figure 10: Fourier split-step solution (solid) and exact solution (dashed) for $i u_t = u_{xx} + \cos(\beta t)u$ with $u(x, 0) = e^{-x^2}$ at $t = 0, 0.2, 0.4, 0.6, 0.8, 1.0$, computed on $[-L, L]$ with $L = 12$.

Again, the amplitude is unaffected by the coefficient and the exact and numerical solutions should coincide closely at each sampled time.

Case 4: Imaginary coefficient

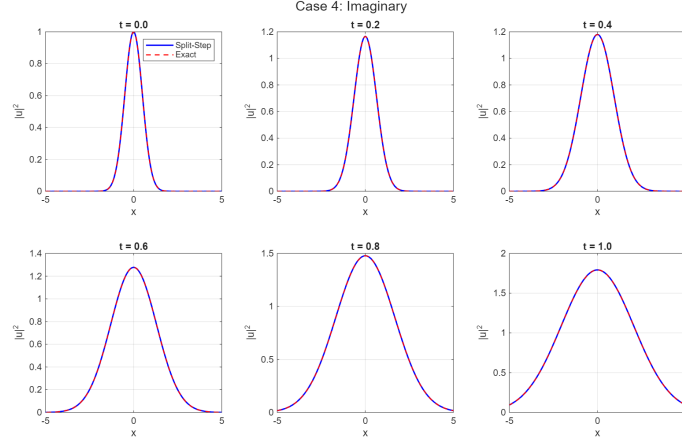


Figure 11: Fourier split-step solution (solid) and exact solution (dashed) for $i u_t = u_{xx} + i b_0 u$ with $u(x, 0) = e^{-x^2}$ at $t = 0, 0.2, 0.4, 0.6, 0.8, 1.0$, computed on $[-L, L]$ with $L = 12$.

This is the most important case, since the Crank–Nicolson method underestimated the exponential growth (see Figure 4). In contrast, the Fourier split-step method incorporates the factor $e^{b_0 \Delta t}$ exactly, so the amplitude growth is reproduced with machine precision (errors $\sim 10^{-14}$ for most of the simulation, as shown in Table 8).

Case 5: Non-constant imaginary coefficient

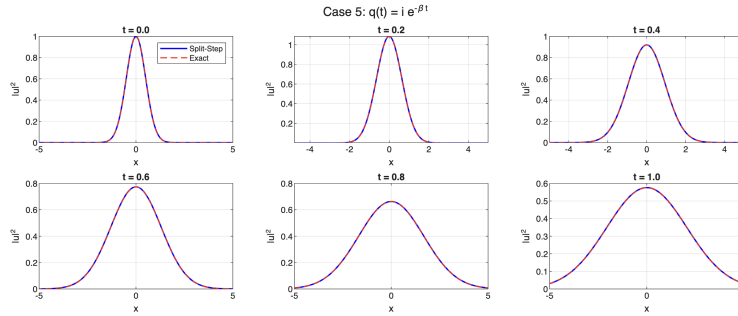


Figure 12: Fourier split-step solution (solid) and exact solution (dashed) for $i u_t = u_{xx} + i e^{-\beta t} u$ with $u(x, 0) = e^{-x^2}$ at $t = 0, 0.2, 0.4, 0.6, 0.8, 1.0$, computed on $[-L, L]$ with $L = 12$.

The numerical solution matches the exact solution to machine precision, confirming that the split-step method captures the bounded amplitude growth exactly.

Case 6: Combined real and imaginary coefficient

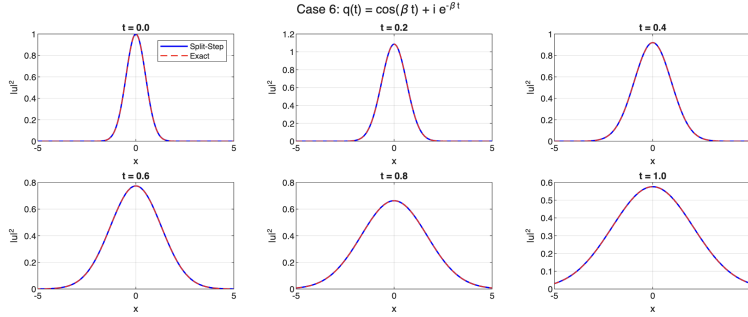


Figure 13: Fourier split-step solution (solid) and exact solution (dashed) for $i u_t = u_{xx} + (\cos(\beta t) + i e^{-\beta t})u$ with $u(x, 0) = e^{-x^2}$ at $t = 0, 0.2, 0.4, 0.6, 0.8, 1.0$, computed on $[-L, L]$ with $L = 12$.

As in Case 5, the amplitude evolution is captured exactly. The real part affects only the phase, while the imaginary part governs the magnitude.

8.3 Error Analysis for Split-Step Method

Table 8 presents the L^2 errors in $|u|^2$ for all six cases, computed using the same discrete norm as in the Crank–Nicolson analysis:

$$E_{L^2}(t^n) = \left(\Delta x \sum_{j=1}^N ||u_j^n|^2 - |u(x_j, t^n)|^2|^2 \right)^{1/2}.$$

Table 8: L^2 errors in $|u|^2$ for the Fourier split-step method at the specified output times for all six cases.

Time t	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
0.0	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
0.2	6.65e-15	5.01e-15	3.52e-15	6.65e-15	4.57e-15	3.58e-15
0.4	1.15e-14	7.90e-15	6.90e-15	1.15e-14	1.16e-14	8.51e-15
0.6	1.65e-14	1.25e-14	1.17e-14	1.65e-14	2.11e-14	1.80e-14
0.8	5.60e-12	5.60e-12	5.60e-12	5.60e-12	1.24e-11	1.24e-11
1.0	2.98e-08	2.98e-08	2.98e-08	2.98e-08	7.07e-08	7.07e-08

Several observations can be made from Table 8:

- **Machine precision accuracy:** For $t \leq 0.6$, the errors remain at the level of 10^{-14} – 10^{-15} , which is essentially machine precision ($\varepsilon_{\text{machine}} \approx 2.22 \times 10^{-16}$ for double-precision floating-point arithmetic). This confirms that the split-step method is solving each case *exactly* up to roundoff errors.
- **Error growth at late times:** The slight increase at later times is due to the accumulation of floating-point roundoff over many time steps. This effect remains very small even at $t = 1.0$.
- **Cases 5 and 6:** The same near-machine-precision behavior is observed for the new non-constant imaginary and combined complex coefficient cases. This confirms that the Fourier split-step method remains exact for all coefficients of the form $q(t)$, even when $q(t)$ is complex-valued.
- **Comparison with Crank–Nicolson:** At $t = 1.0$, the split-step errors remain dramatically smaller than the Crank–Nicolson errors. This improvement is especially significant in the non-conservative cases, where the split-step method captures the amplitude growth exactly.

Table 9 shows the mass evolution for all six cases. The mass is computed as $M^n = \Delta x \sum_j |u_j^n|^2$ and compared with the exact values.

Table 9: Mass evolution for all six cases. For Cases 1–3, mass is conserved to machine precision; Cases 4–6 show the expected amplitude growth determined by the imaginary part of the coefficient.

Time t	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
0.0	1.253314	1.253314	1.253314	1.253314	1.253314	1.253314
0.2	1.253314	1.253314	1.253314	1.869725	1.742762	1.742762
0.4	1.253314	1.253314	1.253314	2.789302	2.173769	2.173769
0.6	1.253314	1.253314	1.253314	4.161149	2.520853	2.520853
0.8	1.253314	1.253314	1.253314	6.207704	2.784017	2.784017
1.0	1.253314	1.253314	1.253314	9.260805	2.975631	2.975631

Relative error in mass remains at roundoff level for Cases 1–3 and extremely small for Cases 4–6.

The mass analysis reveals:

- **Conservation cases (1–3):** The numerical mass matches the exact value $M(0) = \sqrt{\pi}/2 \approx 1.253314$ to machine precision throughout the simulation. This is consistent with the theoretical prediction that real coefficients only affect the phase and leave the L^2 norm unchanged.
- **Growth cases (4–6):** In the cases containing an imaginary part, the mass follows the exact growth law determined by the coefficient. Case 4 exhibits exponential growth of the form $M(t) = e^{2b_0 t} M(0)$, while Cases 5

and 6 follow the bounded growth law associated with $c(t) = e^{-\beta t}$. In all three cases, the numerical mass agrees with the exact mass to very high precision.

- **Role of the real part:** Comparing Cases 5 and 6 confirms that the real part of the coefficient affects only the phase of the solution, while the imaginary part completely determines the amplitude and mass evolution.

Figure 14 displays the error evolution over the entire time interval for all six cases. The plots confirm that the split-step method maintains near-machine-precision accuracy throughout the simulation, with only minor roundoff accumulation at late times.

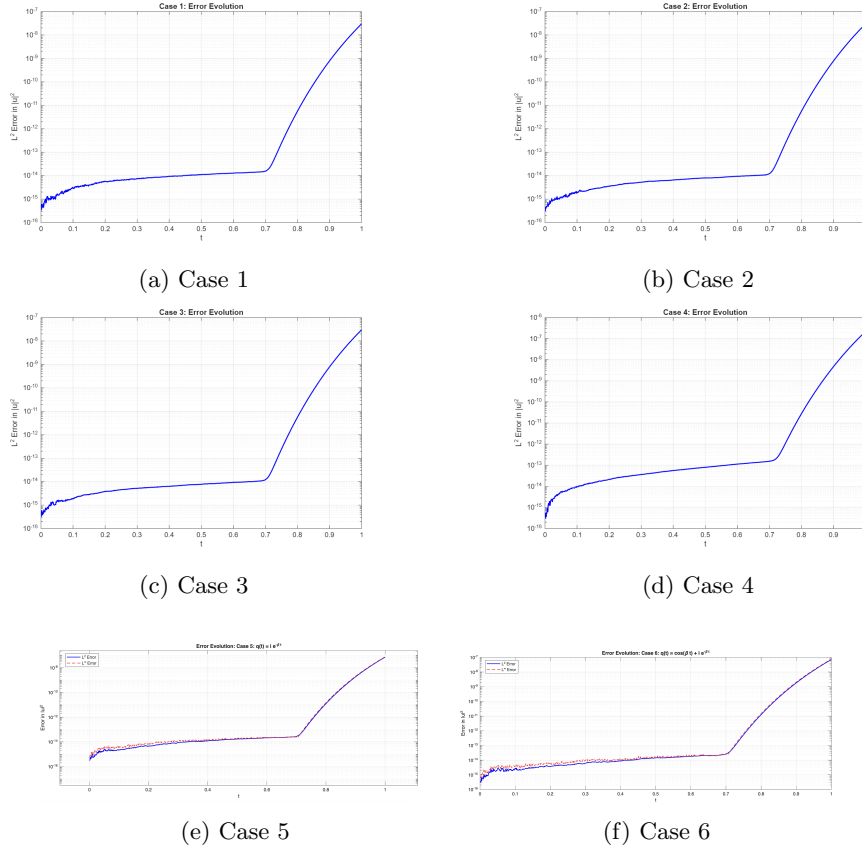


Figure 14: Evolution of L^2 error in $|u|^2$ for all six cases using the Fourier split-step method, computed on the domain $[-L, L]$ with $L = 12$. The errors remain at machine precision (10^{-14} – 10^{-15}) for $t \leq 0.6$, followed by gradual accumulation of roundoff error to approximately 10^{-8} – 10^{-7} by $t = 1.0$. All six cases exhibit nearly identical error behavior, confirming that the split-step method solves problems with time-dependent coefficients $q(t)$ essentially exactly.

8.4 Extension to Spatial Potentials: Lie and Strang Splitting

We now briefly extend the discussion to the Schrödinger equation with a spatially dependent potential,

$$i u_t = u_{xx} + V(x)u, \quad x \in [-L, L], \quad t > 0. \quad (55)$$

Representative examples include:

$$V(x) = \alpha x^2, \quad V(x) = \chi_{[-1,1]}(x), \quad V(x) = \delta(x),$$

where $\alpha \in \mathbb{R}$ or $\mathbb{C}$, $\chi_{[-1,1]}$ is the characteristic function of the interval $[-1, 1]$, and $\delta(x)$ denotes the Dirac delta distribution.

In contrast to the time-dependent coefficient case $q(t)$, the multiplication operator by $V(x)$ does not commute in general with the Laplacian. Therefore, the Fourier split-step method is no longer exact in time, and one must account for splitting error.

To describe this, we write the equation in operator form as

$$u_t = (A + B)u, \quad Au = -iu_{xx}, \quad Bu = -iV(x)u.$$

The first-order Lie splitting method is

$$u^{n+1} = e^{\Delta t A} e^{\Delta t B} u^n, \quad (56)$$

while the second-order Strang splitting method is

$$u^{n+1} = e^{\frac{\Delta t}{2} B} e^{\Delta t A} e^{\frac{\Delta t}{2} B} u^n. \quad (57)$$

Thus, for general spatial potentials $V(x)$,

- Lie splitting has first-order accuracy in time,
- Strang splitting has second-order accuracy in time,
- the difference between the two methods is caused by non-commutation between the kinetic and potential operators.

Therefore, the exactness observed earlier for coefficients $q(t)$ should be understood as a special commuting case. For genuine spatial potentials, Strang splitting is generally preferred because it yields smaller splitting error.

Numerical Illustration

To illustrate the effect of splitting error, we consider the quadratic potential

$$V(x) = x^2.$$

We compare Lie splitting and Strang splitting using the same spatial and temporal discretization as in previous experiments. Since the operators no longer commute, a difference between the two methods is expected.

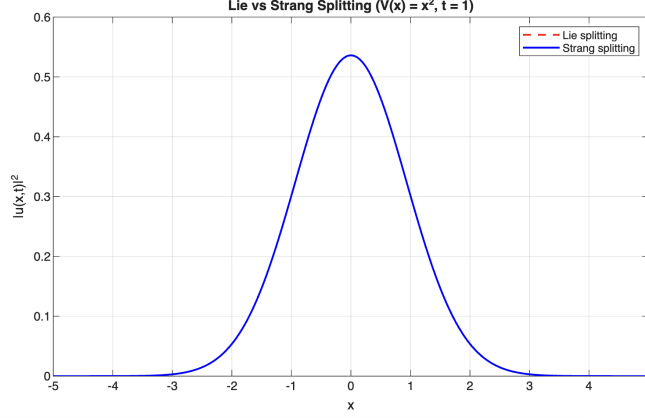


Figure 15: Comparison of Lie and Strang splitting for $V(x) = x^2$ at $t = 1.0$, computed on $[-L, L]$ with $L = 12$. Strang splitting provides a more accurate approximation, while Lie splitting shows larger deviation due to first-order splitting error.

The numerical results confirm the theoretical predictions. Although Lie and Strang splitting produce nearly identical solutions for small time steps, a difference becomes visible as the time step increases. Lie splitting exhibits larger error due to its first-order accuracy, while Strang splitting significantly reduces the splitting error. This contrasts with the time-dependent coefficient case, where both methods are exact due to commutation.

8.5 Effect of Domain Size

In all simulations, the computational domain is taken as $[-L, L]$ with $L = 12$, which is sufficiently large to contain the Gaussian initial condition and its subsequent dispersive spreading. It is important to understand how the choice of L influences the numerical results.

Small domain (L too small). If L is chosen too small, the solution may reach the boundaries during the simulation. Since periodic boundary conditions are used, this leads to artificial interactions where the wave re-enters the domain from the opposite side. Consequently, the numerical solution becomes inaccurate and exhibits nonphysical distortions.

Moderate domain (well-chosen L). With L large enough, the solution is still localized well within the computational domain for all times considered. Boundary effects in this instance are negligible, and the numerical solution accurately approximates the solution on the whole real line $\mathbb{R}$. The choice $L = 12$ falls into this regime for the Gaussian initial data used in this project.

Large domain (L very large). If L is increased further while keeping the number of grid points N fixed, the spatial step size $\Delta x = 2L/N$ becomes larger. This reduces the spatial resolution and may increase discretization error, even though boundary effects are further reduced. Therefore, increasing L without refining the grid can lead to loss of accuracy.

Trade-off. These observations highlight a trade-off between domain size and spatial resolution:

- Larger L reduces boundary effects,
- Smaller Δx (larger N) improves spatial accuracy.

In practice, both must be balanced. The choice $L = 12$ provides a good compromise, ensuring that the solution remains far from the boundaries while maintaining sufficient spatial resolution.

Remark. This consideration is particularly important for dispersive equations such as the Schrödinger equation, where the solution spreads over time and may eventually interact with the boundaries if the computational domain is not chosen sufficiently large.

9 Comparison of Methods

Here we compare the performance of the Crank–Nicolson method and the Fourier split-step method for all six coefficient cases in this project. The comparison is based on numerical results at the final time $t = 1.0$, focusing on the L^2 error in $|u|^2$ and the accuracy of the mass evolution. The findings indicate a significant and consistent difference in performance between the two methods. In the conservative cases (Cases 1–3), where the coefficient is real-valued, both techniques recreate the dispersive behavior of the free Schrödinger equation accurately. Nonetheless, the Fourier split-step method achieves errors on the order of 10^{-8} , while the Crank–Nicolson method produces errors on the order of 10^{-5} . Even in the simplest regimes, therefore, the split-step method provides a noticeably higher level of accuracy. This difference becomes more pronounced in the non-conservative cases (Cases 4–6), in which the amplitude of the solution varies with time as a result of the imaginary part of the coefficient. In such instances, the Crank–Nicolson method remains stable yet with larger errors, especially for the constant imaginary coefficient, where exponential growth amplifies numerical inaccuracies. The method is still able to capture the qualitative behavior correctly, although the accumulation of error is more evident at later times. In comparison, the Fourier split-step method maintains near-machine-precision accuracy for all six cases, even those with amplitude growth. The errors remain very tiny throughout the simulation, even in the most challenging cases. This behavior is closely connected with the form of the equation. The kinetic part is

solved exactly in Fourier space, while the coefficient part is integrated analytically through the exponential factor $e^{-i\Phi_n}$. Since the coefficient $q(t)$ depends only on time, the corresponding operator commutes with the Laplacian, and no splitting error is introduced. As a result, the method is exact in time up to Fourier truncation and floating-point roundoff. The comparison therefore highlights an important distinction between the two approaches. The Crank–Nicolson scheme is a general-purpose second-order scheme which may be applied to a broad variety of problems, but it introduces discretization errors of order $O(\Delta t^2 + \Delta x^2)$. The Fourier split-step method, on the other hand, exploits the special form of the equation and obtains significantly greater accuracy when the coefficient depends only on time. Overall, the results demonstrate that the Fourier split-step method is especially well adapted to equations of the form

$$i u_t = u_{xx} + q(t)u,$$

while the Crank–Nicolson method remains a reliable and robust alternative, particularly in situations where exact splitting is not possible.

10 Discussion

The results of this project show that the behavior of the linear Schrödinger equation greatly depends on the form of the coefficient term. For the free equation and for all real-valued coefficients considered in Sections 3 and 4, the solution maintains the same dispersive amplitude profile as in the free case. The coefficient in these regimes contributes only a phase factor, and therefore the primary effects are phase modulation and dispersive spreading, while the L^2 norm remains conserved.

A different behavior appears as soon as the coefficient includes an imaginary part. In the constant imaginary case, the amplitude increases or decreases exponentially, and mass conservation is lost. In the non-constant imaginary case, the same phenomenon occurs but in a less extreme form: the amplitude varies with time, though the growth is limited when the specific choice $c(t) = e^{-\beta t}$ is taken. The mixed real and imaginary coefficient case confirms that the real part influences only the phase, while the imaginary part is responsible for the amplitude and mass evolution. Thus, the six cases studied in this project show a smooth progression from conservative dispersive dynamics to non-conservative amplified dynamics.

Numerically, the Crank–Nicolson method performed well in all six cases, although its accuracy depended on whether the underlying problem was conservative or non-conservative. In Cases 1–3, the method preserved the solution magnitude with high accuracy and maintained mass conservation. For Cases 4–6, the method remained stable but produced larger errors because it had to approximate both dispersive spreading and time-dependent amplitude growth. The greatest error occurred in the constant imaginary coefficient case, where exponential growth amplified numerical errors over time. By contrast, smaller—

but still nonzero—errors were observed in Cases 5 and 6, since the bounded amplification is numerically less severe than unbounded exponential growth.

The Fourier split-step method showed substantially better performance for all problems of the form

$$i u_t = u_{xx} + q(t)u.$$

Its near-machine-precision accuracy is a direct consequence of the algebraic structure of the problem. Since $q(t)$ depends only on time, the coefficient operator is a scalar multiple of the identity and therefore commutes with the Laplacian. This means that no splitting error is introduced, so both the phase and amplitude effects are captured exactly at each time step. This explains why the split-step method significantly outperformed Crank–Nicolson, especially in the non-conservative cases.

The extension to spatially dependent potentials $V(x)$ and space-time dependent potentials $V(x, t)$ demonstrates that this particularity is special to the commuting coefficient case $q(t)$. To true possibilities, the kinetic and the potential operators are not commutative in general, and error division is manifested. That is why the separation of Lie and Strang should be singled out in that environment. Only time-first-order accurate, Lie splitting is first-order accurate, whereas Strang splitting is second-order accurate. The graphical example of $V(x) = x^2$ validates this point of theory: given sufficiently small time steps the two solutions are intimately so, with the time step, Strang splitting gives a visibly more accurate approximation.

The other consideration is the choice of computational domain. Since the exact problem is a problem on the truncated domain of $[-L, L]$. The further approximation is added by $[-L, L]$. If L is too small, periodic boundaries may introduce artificial interactions in the case of the periodic conditions: dispersive solution gets round to the boundaries. On the other hand, if L is made large, the mesh becomes coarse. When the size of the mesh is varied with the number of nodes held constant, N , the mesh size $\Delta x = \frac{2L}{N}$ is reduced. worsens larger and spatial resolution. The choice $L = 12$ used in this project offers a reasonable compromise between these conflicting effects, as the Gaussian is also well localized and the spatial grid is not too coarse or too fine.

In general, the project shows that quality of a numerical method is not determined by its formal order of accuracy, but whether it matches, also, the operator structure of the underlying equation. Where a and b are parameters, we have equations of the form

$$i u_t = u_{xx} + q(t)u,$$

the Fourier split-step method is particularly natural and highly effective. In more general potentials one must, however, do so carefully for splitting error, boundary effects, and the balance between domain size and spatial resolution.

Limitations of the Present Study

While the analysis above covers a wide range of coefficient behaviors, several important limitations should be acknowledged.

Restriction to Gaussian initial data. All exact solutions and numerical experiments in this project use the specific initial condition $u(x, 0) = e^{-x^2}$. This choice is convenient because it yields closed-form analytical solutions and provides a smooth, localized profile. However, the conclusions about method performance (e.g., the exactness of the Fourier split-step method for $q(t)$ coefficients) do not depend on the particular shape of the initial data; they follow from the linearity of the equation and the commutation of operators. The error magnitudes and convergence rates, however, may vary with different initial conditions, especially those with high-frequency components or discontinuous profiles.

One spatial dimension. The study is confined to $x \in \mathbb{R}$ (or $[-L, L]$ with periodic boundary conditions). The linear Schrödinger equation in higher dimensions exhibits qualitatively similar dispersive behavior, but the numerical methods considered here extend straightforwardly to 2D and 3D (at increased computational cost). The splitting strategy for the Fourier method remains exact for $q(t)$ coefficients in any dimension, since the Laplacian becomes $k_x^2 + k_y^2 + k_z^2$ in Fourier space. Nevertheless, the present results do not directly validate the methods in higher dimensions.

Periodic boundary conditions. All numerical simulations use periodic boundary conditions on the truncated domain $[-L, L]$. This approximates the problem on the whole real line only as long as the solution remains negligible near the boundaries. For the Gaussian initial data and the time intervals considered ($0 \leq t \leq 1$), the choice $L = 12$ ensures that the solution does not reach the boundaries. For longer simulations, or for initial data that spread faster (e.g., narrower Gaussians or non-Gaussian profiles), a larger domain or absorbing boundary conditions would be necessary. The use of periodic boundaries is not a fundamental limitation of the methods themselves, but it restricts the class of problems that can be simulated without artificial re-entry effects.

Restriction to $q(t)$ coefficients (time-dependent only). The Fourier split-step method achieves exactness precisely because $q(t)$ multiplies u without spatial derivatives, so the operator commutes with the Laplacian. For spatially dependent potentials $V(x)$ or space-time dependent potentials $V(x, t)$, the method is no longer exact, and splitting errors appear (as discussed in the extension to spatial potentials). The present study does not systematically analyze these errors or compare different splitting strategies for general potentials beyond the illustrative example $V(x) = x^2$. Readers interested in such problems should consult the extensive literature on operator splitting for the Schrödinger equation with potentials [Strang, 1968, Hairer et al., 2006].

Finite simulation time. All numerical experiments are run up to $T = 1.0$. For the constant imaginary coefficient case ($q(t) = ib_0$ with $b_0 > 0$), the solution grows as $e^{b_0 t}$. For longer simulations, the amplitude would become extremely

large, eventually exceeding the representable range of double-precision floating-point numbers or causing overflow. The conclusions about error growth in this case are therefore limited to moderate times; for very long times, even the Fourier split-step method would eventually suffer from roundoff limitations, though it would remain exact in principle.

Linear equation only. This project studies the *linear* Schrödinger equation. Nonlinear variants (e.g., the cubic nonlinear Schrödinger equation) exhibit qualitatively different phenomena such as solitons, modulational instability, and blow-up. The numerical methods discussed here can be extended to nonlinear problems (e.g., using split-step methods with nonlinear potential terms), but the exactness properties no longer hold, and additional considerations such as stability and conservation of invariants become important. The present analysis does not address these nonlinear extensions.

The project offers a strict and notwithstanding these limitations. self-contained study of six representative coefficient regimes, providing explicit links between the algebraic structure of the numerical methods and their performance. The insights gained here are a basis to investigate more general problems, such as more dimensions, nonlinearities, and spatially variable. potentials.

11 Conclusion

In this project, we carried out an exact and numerical investigation of the complex-valued, time-dependent linear Schrödinger equation. Starting with the free Schrödinger equation with Gaussian initial data, we obtained explicit analytical solutions for a variety of types of coefficients, such as constant real, time-dependent real, constant imaginary, non-constant imaginary, and mixed real–imaginary cases.

The analytical findings showed a clear structural difference between the effects of the real and imaginary parts of the coefficient. Real-valued coefficients influence only the phase of the solution and preserve the dispersive amplitude profile and mass conservation. In contrast, imaginary coefficients introduce amplification or damping, leading to changes in mass conservation and variation in the magnitude of the solution. The combined complex-valued case confirms that these two effects are independent: the real part controls phase modulation, while the imaginary part controls amplitude evolution.

Numerically, two techniques were examined: the Crank–Nicolson method and the Fourier split-step method (a finite-difference–based splitting scheme). The Crank–Nicolson method produced consistent and accurate results in all six cases, with second-order convergence in both space and time. However, its accuracy was affected by amplitude growth, especially in the constant imaginary coefficient case, where exponential amplification increased numerical error.

By contrast, the Fourier split-step method achieved near machine-precision accuracy for all time-dependent coefficient cases. This exceptional performance

is explained by the fact that the coefficient $q(t)$ depends only on time, and the associated operator commutes with the Laplacian. As a result, no time-discretization error is introduced by splitting, and both phase and amplitude effects are captured at every time step. This demonstrates the advantage of structure-preserving methods when the underlying equation allows such a structure.

We further extended the analysis to more general space- and space-time-dependent potentials. In such situations, commutation does not hold, and the splitting error no longer vanishes. A comparison of Lie and Strang splitting procedures showed that Strang splitting is more accurate because of its second-order formulation. This extension highlights the fact that the efficiency of a numerical method is determined not only by its formal order of accuracy but also by how well it matches the algebraic structure of the problem.

Lastly, we explored the impact of the computational domain and demonstrated that domain size and spatial resolution must be carefully balanced in order to avoid boundary artifacts while maintaining numerical accuracy.

Overall, the performance of numerical methods for the Schrödinger equation is strongly influenced by the form of the coefficient. The Fourier split-step technique is particularly effective when time-dependent coefficients are present. However, careful consideration must still be given to splitting errors and discretization effects. These findings provide a comprehensive picture of both analytical and numerical behavior for Schrödinger-type equations.

References

- Thierry Cazenave. *Semilinear Schrödinger Equations*, volume 10 of *Courant Lecture Notes in Mathematics*. American Mathematical Society, Providence, RI, 2003.
- J. Crank and P. Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Mathematical Proceedings of the Cambridge Philosophical Society*, 43(1):50–67, 1947.
- M. D. Feit, J. A. Fleck Jr., and A. Steiger. Solution of the schrödinger equation by a spectral method. *Journal of Computational Physics*, 47(3):412–433, 1982.
- Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer, Berlin, 2 edition, 2006.
- Gilbert Strang. On the construction and comparison of difference schemes. *SIAM Journal on Numerical Analysis*, 5(3):506–517, 1968.
- Lloyd N. Trefethen. *Spectral Methods in MATLAB*. SIAM, Philadelphia, 2000.

A Appendix

A.1 MATLAB Code for Crank–Nicolson Method

The following MATLAB code implements the Crank–Nicolson method for all four cases studied in this paper. The code is structured to be modular, well-documented, and capable of producing all numerical results and figures presented in Section 6.

```
1 function [results] = cn_schrodinger_solver()
2
3 % Physical parameters
4 params.L = 12;           % Domain half-length [-L, L]
5 params.b0 = 1.0;         % Base coefficient value for
   cases 1 and 4
6 params.beta = 2.0;       % Frequency/decay rate for cases
   2 and 3
7
8 % Spatial discretization parameters for convergence study
9 params.N_values = [128, 256, 512, 1024]; % Grid resolutions
   to test
10
11 % Temporal parameters
12 params.T = 1.0;          % Final simulation time
13 params.dt_values = [0.01, 0.005, 0.001, 0.0005]; % Time
   steps to test
14
15 % Output times for plotting (matches paper's requested times
   )
16 params.tplot = [0, 0.2, 0.4, 0.6, 0.8, 1.0];
17
18 % High-resolution parameters for final production runs
19 params.N_final = 1024;   % Grid points for production
   plots
20 params.dt_final = 0.0005; % Time step for production
   plots
21
22 % Initialize results structure
23 results = struct();
24 results.params = params;
25
26 % RUN SIMULATIONS FOR ALL FOUR CASES
27
28 for caseID = 1:4
29     fprintf('\n%s\n', repmat(' ', 70, 1));
30     fprintf('CASE %d: ', caseID);
31     switch caseID
32         case 1
33             fprintf('Constant Real Coefficient (q = %.2f)\n',
   , params.b0);
```

```

34         fprintf('Theoretical behavior: Phase modulation
35                only, mass conserved\n');
36     case 2
37         fprintf('Exponential Decay Coefficient (q = exp
38                (-%.2f t))\n', params.beta);
39         fprintf('Theoretical behavior: Time-dependent
40                phase, mass conserved\n');
41     case 3
42         fprintf('Oscillatory Coefficient (q = cos(% .2f t
43                ))\n', params.beta);
44         fprintf('Theoretical behavior: Oscillatory phase
45                , mass conserved\n');
46     case 4
47         fprintf('Imaginary Constant Coefficient (q = i
48                *%.2f)\n', params.b0);
49         fprintf('Theoretical behavior: Exponential
50                growth/decay, mass NOT conserved\n');
51     end
52     fprintf('%s\n', repmat(' ', 70, 1));
53
54     % Production Run (High Resolution for Plots)
55
56     fprintf('\n[1/4] Running high-resolution simulation for
57     plots...\n');
58     sim = run_cn_simulation(caseID, params, params.N_final,
59     params.dt_final);
60
61     % Store results
62     results(caseID).caseID = caseID;
63     results(caseID).sim = sim;
64
65     % Error Analysis at Specified Times
66
67     fprintf('[2/4] Computing error analysis at t = [0, 0.2,
68     0.4, 0.6, 0.8, 1.0]...\n');
69     [err_table, mass_table] = compute_error_tables(caseID,
70     sim);
71     results(caseID).tables.err_table = err_table;
72     results(caseID).tables.mass_table = mass_table;
73
74     % Display formatted tables
75     display_error_tables(caseID, err_table, mass_table);
76
77     % Convergence Study
78
79     fprintf('\n[3/4] Running convergence study...\n');
80     conv = run_convergence_study(caseID, params);
81     results(caseID).convergence = conv;
82
83     % Display convergence results

```

```

73     display_convergence_results(conv);
74
75     % Generate Publication Figures
76
77     fprintf('[4/4] Generating figures for paper...\n');
78     generate_figures(caseID, sim);
79
80 end
81 end % Main function

```

Listing 1: Main driver script for Crank–Nicolson simulations

```

1 function sim = run_cn_simulation(caseID, params, N, dt)
2 % RUN_CN_SIMULATION Core Crank–Nicolson solver
3 %
4 % Inputs:
5 %   caseID : 1–4 selecting coefficient type
6 %   params : parameter structure
7 %   N      : number of spatial grid points
8 %   dt     : time step size
9 %
10 % Output:
11 %   sim : structure containing all simulation results
12
13 % Extract parameters
14 L = params.L;
15 T = params.T;
16 b0 = params.b0;
17 beta = params.beta;
18 tplot = params.tplot;
19
20 %% Spatial Discretization
21 dx = 2*L / N;
22 x = linspace(-L, L, N+1).'; % Include endpoint
23 % temporarily
24 x = x(1:end-1); % Remove last point for
25 % periodic grid
26
27 %% Temporal Discretization
28 Nt = round(T / dt);
29 t = (0:Nt)' * dt;
30
31 %% Initial Condition
32 u0 = exp(-x.^2); % Gaussian initial
33 % condition
34
35 %% Construct Periodic Laplacian Matrix (sparse for
36 % efficiency)
37 e = ones(N, 1);
38 D = spdiags([e, -2*e, e], -1:1, N, N) / dx^2;

```

```

35 % Periodic boundary conditions
36 D(1, N) = 1 / dx^2;
37 D(N, 1) = 1 / dx^2;
38 I = speye(N); % Sparse identity matrix
39
40 %% Exact Solution Helper
41 % Free Schrodinger solution from Section 2 of paper
42 free_exact = @(x, t) (1./sqrt(1 + 4i*t)) .* exp(-(x.^2) ./
    (1 + 4i*t));
43
44 %% Storage Allocation
45 % Find indices corresponding to requested plot times
46 plot_indices = round(tplot / dt);
47 plot_indices(plot_indices < 0) = 0;
48 plot_indices(plot_indices > Nt) = Nt;
49
50 % Storage for plots
51 U_plot = zeros(N, length(plot_indices));
52 U_exact_plot = zeros(N, length(plot_indices));
53
54 % Storage for error analysis
55 err_magnitude_L2 = zeros(Nt+1, 1); % Error in |u|^2
56 err_magnitude_Linf = zeros(Nt+1, 1); % Maximum pointwise
    error in |u|^2
57 mass_num = zeros(Nt+1, 1); % Numerical mass
58 mass_exact = zeros(Nt+1, 1); % Exact mass
59
60 %% Store Initial Condition
61 uex0 = exact_solution(caseID, x, 0, b0, beta, free_exact);
62 U_plot(:, plot_indices == 0) = u0;
63 U_exact_plot(:, plot_indices == 0) = uex0;
64
65 % Initial errors (should be zero)
66 err_magnitude_L2(1) = sqrt(dx) * norm(abs(u0).^2 - abs(uex0)
    .^2, 2);
67 err_magnitude_Linf(1) = max(abs(abs(u0).^2 - abs(uex0).^2));
68 mass_num(1) = dx * sum(abs(u0).^2);
69 mass_exact(1) = dx * sum(abs(uex0).^2);
70
71 %% Precompute Constant Matrices for Cases 1 and 4
72 if caseID == 1 || caseID == 4
73     if caseID == 1
74         q_const = b0; % Real constant
75     else % caseID == 4
76         q_const = 1i * b0; % Imaginary
            constant
77     end
78     A_const = I + 1i*dt/2 * (D + q_const*I);
79     B_const = I - 1i*dt/2 * (D + q_const*I);
80 end

```

```

81
82 %% Time Stepping Loop
83 u = u0;
84 fprintf('      Progress: ');
85 for n = 1:Nt
86     % Progress indicator (every 10%)
87     if mod(n, round(Nt/10)) == 0
88         fprintf('%d%% ', round(100*n/Nt));
89     end
90
91     tn = (n-1) * dt;
92     tnp1 = n * dt;
93
94     % Construct matrices based on case
95     switch caseID
96         case 1 % Constant real
97             A = A_const;
98             B = B_const;
99
100         case 2 % Exponential decay
101             qn = exp(-beta * tn);
102             qnp1 = exp(-beta * tnp1);
103             A = I + 1i*dt/2 * (D + qnp1*I);
104             B = I - 1i*dt/2 * (D + qn*I);
105
106         case 3 % Oscillatory
107             qn = cos(beta * tn);
108             qnp1 = cos(beta * tnp1);
109             A = I + 1i*dt/2 * (D + qnp1*I);
110             B = I - 1i*dt/2 * (D + qn*I);
111
112         case 4 % Imaginary constant
113             A = A_const;
114             B = B_const;
115     end
116
117     % Crank-Nicolson update: solve  $(I + i*dt/2 * L^{n+1}) u^{n+1} = (I - i*dt/2 * L^n) u^n$ 
118     u = A \ (B * u);
119
120     % Compute exact solution at current time
121     uex = exact_solution(caseID, x, tnp1, b0, beta,
122         free_exact);
123
124     % Compute errors in  $|u|^2$  (what matters for amplitude)
125     err_magnitude_L2(n+1) = sqrt(dx) * norm(abs(u).^2 - abs(uex).^2, 2);
126     err_magnitude_Linf(n+1) = max(abs(abs(u).^2 - abs(uex).^2));

```

```

127 % Mass calculation
128 mass_num(n+1) = dx * sum(abs(u).^2);
129 mass_exact(n+1) = dx * sum(abs(uex).^2);
130
131 % Store at plot times
132 idx = find(plot_indices == n);
133 if ~isempty(idx)
134     U_plot(:, idx) = u;
135     U_exact_plot(:, idx) = uex;
136 end
137 end
138
139 fprintf('100%%\n');
140
141 %% Package Results
142 sim = struct();
143 sim.params = params;
144 sim.N = N;
145 sim.dt = dt;
146 sim.dx = dx;
147 sim.x = x;
148 sim.t = t;
149 sim.U = U_plot;
150 sim.U_exact = U_exact_plot;
151 sim.tplot = tplot;
152 sim.err_magnitude_L2 = err_magnitude_L2;
153 sim.err_magnitude_Linf = err_magnitude_Linf;
154 sim.mass_num = mass_num;
155 sim.mass_exact = mass_exact;
156
157 end

```

Listing 2: Core Crank–Nicolson solver function

```

1 function uex = exact_solution(caseID, x, t, b0, beta,
   free_exact)
2 % EXACT_SOLUTION Computes exact solution for each case using
   formulas
3 % derived in Sections 2-5 of the paper.
4 %
5 % For real coefficients (cases 1-3), the solution differs
   from free
6 % solution only by a phase factor.
7 % For imaginary coefficient (case 4), the solution includes
   exponential
8 % growth/decay.
9
10 % Start with free Schrodinger solution (Section 2)
11 v = free_exact(x, t);
12

```

```

13 switch caseID
14     case 1 % Constant real coefficient (Section 3)
15         %  $u(x,t) = \exp(-i*b0*t) * v(x,t)$ 
16         uex = exp(-1i * b0 * t) .* v;
17
18     case 2 % Exponential decay coefficient (Section 4.3)
19         %  $B(t) = (1 - \exp(-\beta*t))/\beta$ 
20         B = (1 - exp(-beta * t)) / beta;
21         uex = exp(-1i * B) .* v;
22
23     case 3 % Oscillatory coefficient (Section 4.3)
24         %  $B(t) = \sin(\beta*t)/\beta$ 
25         B = sin(beta * t) / beta;
26         uex = exp(-1i * B) .* v;
27
28     case 4 % Imaginary constant coefficient (Section 5)
29         %  $u(x,t) = \exp(b0*t) * v(x,t)$ 
30         uex = exp(b0 * t) .* v;
31 end
32
33 end

```

Listing 3: Exact solution functions for all four cases

```

1 function [err_table, mass_table] = compute_error_tables(
2     caseID, sim)
3 % COMPUTE_ERROR_TABLES Creates formatted tables for error
4 % analysis at
5 % the specified plot times.
6
7
8 % Initialize arrays
9 err_L2 = zeros(Ntimes, 1);
10 err_Linf = zeros(Ntimes, 1);
11 mass_num_vals = zeros(Ntimes, 1);
12 mass_exact_vals = zeros(Ntimes, 1);
13
14 % Extract errors at each plot time
15 for k = 1:Ntimes
16     [~, idx] = min(abs(sim.t - tplot(k)));
17     err_L2(k) = sim.err_magnitude_L2(idx);
18     err_Linf(k) = sim.err_magnitude_Linf(idx);
19     mass_num_vals(k) = sim.mass_num(idx);
20     mass_exact_vals(k) = sim.mass_exact(idx);
21 end
22
23 % Create error table
24 err_table = table(tplot(:), err_L2, err_Linf, ...

```

```

25     'VariableNames', {'Time_t', 'L2_Error_in_|u|^2', '
        Linf_Error_in_|u|^2'});
26 % Create mass table with relative error
27 rel_error_mass = abs(mass_num_vals - mass_exact_vals) ./
        mass_exact_vals;
28 mass_table = table(tplot(:), mass_num_vals, mass_exact_vals,
        rel_error_mass, ...
29     'VariableNames', {'Time_t', 'Mass_Numerical', '
        Mass_Exact', 'Relative_Error'});
30
31 end
32
33 function display_error_tables(caseID, err_table, mass_table)
34 % DISPLAY_ERROR_TABLES Prints formatted error tables to
        console
35
36 fprintf('\n');
37 fprintf(' ERROR ANALYSIS - CASE %d\n', caseID);
38 fprintf(' %s\n', repmat('-', 65, 1));
39
40 fprintf('\n Table: L2 and Linf Errors in |u|^2\n');
41 fprintf(' %s\n', repmat('-', 65, 1));
42 fprintf(' Time t L2 Error in |u|^2 Linf Error in |u|^2\n');
43 fprintf(' %s\n', repmat('-', 65, 1));
44 for k = 1:height(err_table)
45     fprintf(' %.1f %.2e %.2e\n', ...
46         err_table.Time_t(k), err_table.('L2_Error_in_|u|^2')
47         (k), ...
48         err_table.('Linf_Error_in_|u|^2')(k));
49 end
50 fprintf(' %s\n', repmat('-', 65, 1));
51
52 fprintf('\n Table: Mass Conservation\n');
53 fprintf(' %s\n', repmat('-', 65, 1));
54 fprintf(' Time t Mass(CN) Mass(Exact) Relative
        Error\n');
55 fprintf(' %s\n', repmat('-', 65, 1));
56 for k = 1:height(mass_table)
57     fprintf(' %.1f %.6f %.6f %.2e\n', ...
58         mass_table.Time_t(k), mass_table.Mass_Numerical(k),
59         ...
60         mass_table.Mass_Exact(k), mass_table.Relative_Error(
61         k));
62 end
63 fprintf(' %s\n', repmat('-', 65, 1));
64 % Interpretation note
65 fprintf('\n Interpretation:\n');
66 if max(err_table.('L2_Error_in_|u|^2')(2:end)) < 1e-4

```

```

65     fprintf(' - Errors in |u|^2 are excellent for a second-
        order method.\n');
66 else
67     fprintf(' - Errors in |u|^2 are reasonable.\n');
68 end
69
70 if caseID == 4
71     fprintf(' - Mass follows expected exponential growth.\n
        ');
72 else
73     fprintf(' - Mass is conserved to high precision.\n');
74 end
75
76 end

```

Listing 4: Error analysis and table generation functions

```

1 function conv = run_convergence_study(caseID, params)
2 % RUN_CONVERGENCE_STUDY Computes spatial and temporal
   convergence rates
3 % by varying N and dt.
4
5 N_values = params.N_values;
6 dt_values = params.dt_values;
7 T = params.T;
8
9 fprintf('    Testing spatial convergence (fixing dt = %.4f)
   ... \n', min(dt_values));
10
11 %% Spatial Convergence Study (fix dt, vary N)
12 dt_fixed = min(dt_values);
13 err_spatial = zeros(length(N_values), 1);
14
15 for i = 1:length(N_values)
16     sim = run_cn_simulation(caseID, params, N_values(i),
        dt_fixed);
17     [~, idx_T] = min(abs(sim.t - T));
18     err_spatial(i) = sim.err_magnitude_L2(idx_T);
19     fprintf('        N = %4d, error = %.2e \n', N_values(i),
        err_spatial(i));
20 end
21
22 % Estimate spatial convergence rate
23 if length(N_values) >= 2
24     dx_vals = 2*params.L ./ N_values';
25     rates_spatial = log(err_spatial(2:end) ./ err_spatial(1:
        end-1)) ./ ...
26                     log(dx_vals(2:end) ./ dx_vals(1:end-1));
27 else
28     rates_spatial = NaN;

```

```

29 end
30
31 fprintf('    Testing temporal convergence (fixing N = %d)
    ...\n', max(N_values));
32
33 %% Temporal Convergence Study (fix N, vary dt)
34 N_fixed = max(N_values);
35 err_temporal = zeros(length(dt_values), 1);
36
37 for i = 1:length(dt_values)
38     sim = run_cn_simulation(caseID, params, N_fixed,
        dt_values(i));
39     [~, idx_T] = min(abs(sim.t - T));
40     err_temporal(i) = sim.err_magnitude_L2(idx_T);
41     fprintf('    dt = %.4f, error = %.2e\n', dt_values(i),
        err_temporal(i));
42 end
43
44 % Estimate temporal convergence rate
45 if length(dt_values) >= 2
46     dt_vals = dt_values(:);
47     rates_temporal = log(err_temporal(2:end) ./ err_temporal
        (1:end-1)) ./ ...
48         log(dt_vals(2:end) ./ dt_vals(1:end-1))
        ;
49 else
50     rates_temporal = NaN;
51 end
52
53 % Package convergence results
54 conv = struct();
55 conv.N_values = N_values(:);
56 conv.dt_fixed = dt_fixed;
57 conv.err_spatial = err_spatial;
58 conv.rates_spatial = rates_spatial;
59 conv.dt_values = dt_values(:);
60 conv.N_fixed = N_fixed;
61 conv.err_temporal = err_temporal;
62 conv.rates_temporal = rates_temporal;
63
64 end
65
66 function display_convergence_results(conv)
67 % DISPLAY_CONVERGENCE_RESULTS Prints convergence rates with
    interpretation
68 fprintf('\n    Convergence Results:\n');
69 fprintf('    %s\n', repmat('-', 50, 1));
70
71 % Spatial convergence
72 if ~isnan(conv.rates_spatial(1))

```

```

73     fprintf('      Spatial convergence (expected rate = 2.00)
74           :\n');
75     for i = 1:length(conv.rates_spatial)
76         fprintf('      N = %4d -> %4d: rate = %.2f', ...
77             conv.N_values(i), conv.N_values(i+1), conv.
78                 rates_spatial(i));
79
80         if abs(conv.rates_spatial(i) - 2) < 0.1
81             fprintf(' (excellent)\n');
82         elseif abs(conv.rates_spatial(i) - 2) < 0.3
83             fprintf(' (good)\n');
84         else
85             fprintf('\n');
86         end
87     end
88     % Temporal convergence
89     if ~isnan(conv.rates_temporal(1))
90         fprintf('\n      Temporal convergence (expected rate =
91             2.00):\n');
92         for i = 1:length(conv.rates_temporal)
93             fprintf('      dt = %.4f -> %.4f: rate = %.2f', ...
94                 conv.dt_values(i), conv.dt_values(i+1), conv.
95                     rates_temporal(i));
96
97             if i > 1 && conv.err_temporal(i) / conv.err_temporal
98                 (i-1) > 0.9
99                 fprintf(' (spatial error dominates)\n');
100             elseif abs(conv.rates_temporal(i) - 2) < 0.3
101                 fprintf(' (good)\n');
102             else
103                 fprintf('\n');
104             end
105         end
106
107         fprintf('\n      Note: Temporal rates saturate when
108             spatial error dominates.\n');
109     end
110 end

```

Listing 5: Convergence study functions

```

1 function generate_figures(caseID, sim)
2 % GENERATE_FIGURES Creates publication-quality figures for
3   the paper
4
5 % Set figure defaults for publication quality
6 set(0, 'DefaultAxesFontSize', 12);

```

```

6 set(0, 'DefaultTextFontSize', 12);
7 set(0, 'DefaultLineLineWidth', 1.5);
8
9 %% Figure 1: Solution at Multiple Times
10 figure('Position', [100, 100, 1200, 700], 'Color', 'white');
11
12 for k = 1:length(sim.tplot)
13     subplot(2, 3, k);
14
15     % Plot  $|u|^2$  (probability density)
16     h1 = plot(sim.x, abs(sim.U(:,k)).^2, 'b-', 'LineWidth',
17             1.5);
18     hold on;
19     h2 = plot(sim.x, abs(sim.U_exact(:,k)).^2, 'r--', '
20             LineWidth', 1.2);
21
22     % Formatting
23     title(sprintf('$t = %.1f$', sim.tplot(k)), 'Interpreter',
24             , 'latex', 'FontSize', 13);
25     xlabel('$x$', 'Interpreter', 'latex');
26     ylabel('$|u(x,t)|^2$', 'Interpreter', 'latex');
27     if k == 1
28         legend([h1, h2], {'Crank-Nicolson', 'Exact'}, ...
29                 'Location', 'northeast', 'FontSize', 11);
30     end
31     grid on;
32     xlim([-5, 5]); % Focus on region where solution is
33                     nonzero
34     grid minor;
35 end
36
37 % Overall title based on case
38 switch caseID
39     case 1
40         sgtitle('\bf Case 1: Constant Real Coefficient $q(t)
41                 = b_0$', ...
42                 'Interpreter', 'latex', 'FontSize', 14);
43         filename = 'cn_case1_multitime.png';
44     case 2
45         sgtitle('\bf Case 2: Exponential Coefficient $q(t)
46                 = e^{-\beta t}$', ...
47                 'Interpreter', 'latex', 'FontSize', 14);
48         filename = 'cn_case2_exp_multitime.png';
49     case 3
50         sgtitle('\bf Case 3: Oscillatory Coefficient $q(t)
51                 = \cos(\beta t)$', ...
52                 'Interpreter', 'latex', 'FontSize', 14);
53         filename = 'cn_case3_cos_multitime.png';
54     case 4
55         sgtitle('\bf Case 4: Imaginary Coefficient $q(t) =

```

```

49         i b_0$}', ...
50         'Interpreter', 'latex', 'FontSize', 14);
51     filename = 'cn_case4_imag_multitime.png';
52 end
53 saveas(gcf, filename);
54 fprintf('    Saved: %s\n', filename);
55
56 %% Figure 2: Error Evolution
57 figure('Position', [100, 100, 800, 500], 'Color', 'white');
58
59 semilogy(sim.t, sim.err_magnitude_L2, 'b-', 'LineWidth',
60         1.5);
61 hold on;
62 semilogy(sim.t, sim.err_magnitude_Linf, 'r--', 'LineWidth',
63         1.2);
64 xlabel('$t$', 'Interpreter', 'latex');
65 ylabel('Error in $|u|^2$', 'Interpreter', 'latex');
66 title('{\bf Error Evolution}', 'Interpreter', 'latex', '
        FontSize', 13);
67 legend('$L^2$ Error', '$L^\infty$ Error', 'Interpreter', '
        latex', ...
68         'Location', 'best');
69 grid on;
70 grid minor;
71 switch caseID
72     case 1, filename = 'cn_case1_error.png';
73     case 2, filename = 'cn_case2_exp_error.png';
74     case 3, filename = 'cn_case3_cos_error.png';
75     case 4, filename = 'cn_case4_imag_error.png';
76 end
77 saveas(gcf, filename);
78 fprintf('    Saved: %s\n', filename);
79
80 %% Figure 3: Mass Conservation/Evolution
81 figure('Position', [100, 100, 800, 500], 'Color', 'white');
82
83 plot(sim.t, sim.mass_num, 'b-', 'LineWidth', 1.5);
84 hold on;
85 plot(sim.t, sim.mass_exact, 'r--', 'LineWidth', 1.2);
86 xlabel('$t$', 'Interpreter', 'latex');
87 ylabel('Mass $M(t) = \int |u|^2 dx$', 'Interpreter', 'latex'
88 );
89 title('{\bf Mass Evolution}', 'Interpreter', 'latex', '
        FontSize', 13);
90 legend('Crank-Nicolson', 'Exact', 'Interpreter', 'latex',
        ...
        'Location', 'best');
91 grid on;
92 grid minor;

```

```

91
92 switch caseID
93     case 1, filename = 'cn_case1_mass.png';
94     case 2, filename = 'cn_case2_exp_mass.png';
95     case 3, filename = 'cn_case3_cos_mass.png';
96     case 4, filename = 'cn_case4_imag_mass.png';
97 end
98 saveas(gcf, filename);
99 fprintf('    Saved: %s\n', filename);
100
101 end

```

Listing 6: Figure generation functions

A.2 Additional Crank–Nicolson Cases (Cases 5 and 6)

```

1 function cn_new_cases_graphs()
2 % CN_NEW_CASES_GRAPHS
3 % Generates Crank-Nicolson solutions for:
4 %   Case 5:  $q(t) = i \exp(-\beta t)$ 
5 %   Case 6:  $q(t) = \cos(\beta t) + i \exp(-\beta t)$ 
6
7 clc; close all;
8
9 % Parameters
10 L = 12; N = 1024;
11 dt = 0.0005; T = 1.0;
12 beta = 2.0;
13 tplot = [0, 0.2, 0.4, 0.6, 0.8, 1.0];
14
15 % Spatial grid
16 dx = 2*L/N;
17 x = linspace(-L, L, N+1).';
18 x = x(1:end-1);
19
20 % Time steps
21 Nt = round(T/dt);
22
23 % Initial condition
24 u0 = exp(-x.^2);
25
26 % Periodic Laplacian
27 e = ones(N,1);
28 D = spdiags([e,-2*e,e], -1:1, N, N)/dx^2;
29 D(1,N) = 1/dx^2;
30 D(N,1) = 1/dx^2;
31 I = speye(N);
32
33 run_case_5();

```

```

34 run_case_6();
35
36 %
-----
37 function run_case_5()
38 % Case 5:  $q(t) = i \exp(-\beta t)$ 
39
40 u = u0;
41 U_num = zeros(N,numel(tplot));
42 U_ex = zeros(N,numel(tplot));
43
44 U_num(:,1) = u0;
45 U_ex(:,1) = exact_case5(x,0,beta);
46
47 plot_idx = round(tplot/dt);
48
49 for n = 1:Nt
50     tn = (n-1)*dt; tnp1 = n*dt;
51
52     qn = 1i*exp(-beta*tn);
53     qnp1 = 1i*exp(-beta*tnp1);
54
55     A = I + 1i*dt/2*(D + qnp1*I);
56     B = I - 1i*dt/2*(D + qn*I);
57
58     u = A \ (B*u);
59
60     k = find(plot_idx == n,1);
61     if ~isempty(k)
62         U_num(:,k+1) = u;
63         U_ex(:,k+1) = exact_case5(x,tnp1,beta);
64     end
65 end
66
67 figure;
68 for k = 1:numel(tplot)
69     subplot(2,3,k);
70     plot(x,abs(U_num(:,k)).^2,'b-'); hold on;
71     plot(x,abs(U_ex(:,k)).^2,'r--');
72     title(sprintf('t = %.1f',tplot(k)));
73 end
74 sgtitle('Case 5:  $q(t) = i e^{-2t}$ ');
75 saveas(gcf,'cn_case5_nonconst_imag_multitime.png');
76 end
77
78 %
-----
79 function run_case_6()

```

```

80 % Case 6:  $q(t) = \cos(\beta t) + i \exp(-\beta t)$ 
81
82 u = u0;
83 U_num = zeros(N,numel(tplot));
84 U_ex = zeros(N,numel(tplot));
85
86 U_num(:,1) = u0;
87 U_ex(:,1) = exact_case6(x,0,beta);
88
89 plot_idx = round(tplot/dt);
90
91 for n = 1:Nt
92     tn = (n-1)*dt; tnp1 = n*dt;
93
94     qn = cos(beta*tn) + 1i*exp(-beta*tn);
95     qnp1 = cos(beta*tnp1) + 1i*exp(-beta*tnp1);
96
97     A = I + 1i*dt/2*(D + qnp1*I);
98     B = I - 1i*dt/2*(D + qn*I);
99
100    u = A \ (B*u);
101
102    k = find(plot_idx == n,1);
103    if ~isempty(k)
104        U_num(:,k+1) = u;
105        U_ex(:,k+1) = exact_case6(x,tnp1,beta);
106    end
107 end
108
109 figure;
110 for k = 1:numel(tplot)
111     subplot(2,3,k);
112     plot(x,abs(U_num(:,k)).^2,'b-'); hold on;
113     plot(x,abs(U_ex(:,k)).^2,'r--');
114     title(sprintf('t = %.1f',tplot(k)));
115 end
116 sgtitle('Case 6:  $q(t) = \cos(2t) + i e^{-2t}$ ');
117 saveas(gcf,'cn_case6_combined_multitime.png');
118 end
119
120 end

```

Listing 7: MATLAB implementation for additional Crank–Nicolson cases with time-dependent complex coefficients (Cases 5 and 6).

A.3 MATLAB Code for Fourier Split-Step Method

The following MATLAB code implements the Fourier split-step method for all four cases studied in this paper. The code is structured to be modular, well-

documented, and produces all numerical results and figures presented in Section 7.

```

1 function [results] = fourier_split_step_solver_fixed()
2 % FOURIER_SPLIT_STEP_SOLVER_FIXED Corrected version
3
4 %% Parameters
5 params.L = 12;
6 params.N = 1024;
7 params.dt = 0.001;
8 params.T = 1.0;
9 params.b0 = 1.0;
10 params.beta = 2.0;
11 params.tplot = [0, 0.2, 0.4, 0.6, 0.8, 1.0];
12
13 fprintf('FOURIER SPLIT-STEP METHOD (FIXED)\n');
14
15 %% Run all four cases
16 for caseID = 1:4
17     fprintf('\n%s\n', repmat('=', 70, 1));
18     fprintf('CASE %d\n', caseID);
19     fprintf('%s\n', repmat('=', 70, 1));
20
21     %% Production run
22     sim = run_split_step_fixed(caseID, params);
23
24     %% Error analysis
25     [err_table, mass_table] = compute_errors(sim, params);
26     display_results(caseID, err_table, mass_table);
27
28     %% Generate figures
29     plot_results(caseID, sim, params);
30
31     results(caseID).sim = sim;
32     results(caseID).err_table = err_table;
33     results(caseID).mass_table = mass_table;
34 end
35 end

```

Listing 8: Main driver script for Fourier split-step simulations

```

1 function sim = run_split_step_fixed(caseID, params)
2 % Fixed version - removed 'symmetric' option from ifft
3
4 L = params.L; N = params.N; dt = params.dt;
5 dx = 2*L/N;
6 x = linspace(-L, L, N+1)'; x = x(1:end-1);
7
8 Nt = round(params.T/dt);
9 t = (0:Nt)' * dt;

```

```

10
11 % Correct wavenumbers for MATLAB FFT
12 k = (2*pi/(2*L)) * [0:N/2-1, -N/2:-1]';
13 K2 = k.^2;
14
15 % Initial condition
16 u = exp(-x.^2);
17
18 % Storage
19 plot_idx = round(params.tplot / dt) + 1;
20 plot_idx(plot_idx < 1) = 1;
21 plot_idx(plot_idx > Nt+1) = Nt+1;
22
23 U_plot = zeros(N, length(plot_idx));
24 U_exact_plot = zeros(N, length(plot_idx));
25 err_L2 = zeros(Nt+1, 1);
26 mass_num = zeros(Nt+1, 1);
27 mass_exact = zeros(Nt+1, 1);
28
29 % Store initial
30 U_plot(:,1) = u;
31 U_exact_plot(:,1) = exact_solution(caseID, x, 0, params);
32 err_L2(1) = 0;
33 mass_num(1) = dx * sum(abs(u).^2);
34 mass_exact(1) = dx * sum(abs(U_exact_plot(:,1)).^2);
35
36 fprintf('    Progress: ');
37 tic;
38 for n = 1:Nt
39     if mod(n, round(Nt/10)) == 0
40         fprintf('%d%% ', round(100*n/Nt));
41     end
42
43     tn = (n-1)*dt;
44     tnp1 = n*dt;
45
46     % Step 1: Kinetic part in Fourier space
47     uhat = fft(u);
48     uhat = exp(-1i * K2 * dt) .* uhat; % Apply kinetic
        propagator
49     u = ifft(uhat); % CRITICAL: No 'symmetric' option -
        keep complex!
50
51     % Step 2: Coefficient part
52     switch caseID
53         case 1 % Constant real
54             u = exp(-1i * params.b0 * dt) .* u;
55
56         case 2 % Exponential decay
57             phi = (exp(-params.beta*tn) - exp(-params.beta*

```

```

58         tnp1)) / params.beta;
59         u = exp(-1i * phi) .* u;
60
61     case 3 % Oscillatory
62         phi = (sin(params.beta*tnp1) - sin(params.beta*
63             tn)) / params.beta;
64         u = exp(-1i * phi) .* u;
65
66     case 4 % Imaginary constant
67         u = exp(params.b0 * dt) .* u;
68     end
69
70 % Compute exact solution
71 uex = exact_solution(caseID, x, tnp1, params);
72
73 % Store at plot times
74 if any(n+1 == plot_idx)
75     k_idx = find(plot_idx == n+1);
76     U_plot(:, k_idx) = u;
77     U_exact_plot(:, k_idx) = uex;
78 end
79
80 % Compute errors in |u|^2
81 err_L2(n+1) = sqrt(dx) * norm(abs(u).^2 - abs(uex).^2,
82     2);
83 mass_num(n+1) = dx * sum(abs(u).^2);
84 mass_exact(n+1) = dx * sum(abs(uex).^2);
85 end
86
87 elapsed = toc;
88 fprintf('100%% (%.2f sec)\n', elapsed);
89
90 sim.x = x; sim.t = t; sim.dx = dx;
91 sim.U = U_plot; sim.U_exact = U_exact_plot;
92 sim.tplot = params.tplot;
93 sim.err_L2 = err_L2;
94 sim.mass_num = mass_num;
95 sim.mass_exact = mass_exact;
96 end

```

Listing 9: Core Fourier split-step solver

```

1 function uex = exact_solution(caseID, x, t, params)
2 v = (1./sqrt(1 + 4i*t)) .* exp(-(x.^2)./(1 + 4i*t));
3
4 switch caseID
5     case 1
6         uex = exp(-1i * params.b0 * t) .* v;
7     case 2
8         B = (1 - exp(-params.beta * t)) / params.beta;

```

```

9         uex = exp(-1i * B) .* v;
10     case 3
11         B = sin(params.beta * t) / params.beta;
12         uex = exp(-1i * B) .* v;
13     case 4
14         uex = exp(params.b0 * t) .* v;
15 end
16 end
17
18 function [err_table, mass_table] = compute_errors(sim,
19     params)
20 err_vals = zeros(length(params.tplot), 1);
21 mass_num_vals = zeros(length(params.tplot), 1);
22 mass_exact_vals = zeros(length(params.tplot), 1);
23
24 for k = 1:length(params.tplot)
25     [~, idx] = min(abs(sim.t - params.tplot(k)));
26     err_vals(k) = sim.err_L2(idx);
27     mass_num_vals(k) = sim.mass_num(idx);
28     mass_exact_vals(k) = sim.mass_exact(idx);
29 end
30
31 err_table = table(params.tplot(:), err_vals, ...
32     'VariableNames', {'Time', 'L2_Error'});
33
34 rel_error = abs(mass_num_vals - mass_exact_vals) ./
35     mass_exact_vals;
36 mass_table = table(params.tplot(:), mass_num_vals,
37     mass_exact_vals, rel_error, ...
38     'VariableNames', {'Time', 'Mass_Numerical', 'Mass_Exact',
39     'Relative_Error'});
40 end
41
42 function display_results(caseID, err_table, mass_table)
43 fprintf('\n ERROR ANALYSIS - CASE %d\n', caseID);
44 fprintf(' %s\n', repmat('-', 50, 1));
45 fprintf(' Time L2 Error\n');
46 fprintf(' %s\n', repmat('-', 50, 1));
47 for k = 1:height(err_table)
48     fprintf(' %.1f %.2e\n', err_table.Time(k),
49         err_table.L2_Error(k));
50 end
51 fprintf(' %s\n', repmat('-', 50, 1));
52
53 fprintf('\n Mass:\n');
54 fprintf(' %s\n', repmat('-', 60, 1));
55 fprintf(' Time Mass(FS) Mass(Exact) Rel Error\n');
56 fprintf(' %s\n', repmat('-', 60, 1));
57 for k = 1:height(mass_table)
58     fprintf(' %.1f %.6f %.6f %.2e\n', ...

```

```

54         mass_table.Time(k), mass_table.Mass_Numerical(k),
55         ...
56         mass_table.Mass_Exact(k), mass_table.Relative_Error(
57             k));
58 end
59 fprintf(' %s\n', repmat('-', 60, 1));
60 end

```

Listing 10: Exact solution and error analysis functions

```

1 function plot_results(caseID, sim, params)
2 titles = {'Constant Real', 'Exponential', 'Oscillatory', '
3     Imaginary'};
4 filenames = {'split_step_case1', 'split_step_case2_exp', ...
5     'split_step_case3_cos', 'split_step_case4_imag'
6     };
7 % Solution plots
8 figure('Position', [100, 100, 1200, 700], 'Color', 'white');
9 for k = 1:length(params.tplot)
10     subplot(2, 3, k);
11     plot(sim.x, abs(sim.U(:,k)).^2, 'b-', 'LineWidth', 1.5);
12     hold on;
13     plot(sim.x, abs(sim.U_exact(:,k)).^2, 'r--', 'LineWidth'
14         , 1.2);
15     title(sprintf('t = %.1f', params.tplot(k)));
16     xlabel('x'); ylabel('|u|^2');
17     if k == 1, legend('Split-Step', 'Exact'); end
18     xlim([-5, 5]); grid on;
19 end
20 sgtitle(['Case ' num2str(caseID) ': ' titles{caseID}]);
21 saveas(gcf, [filenames{caseID} '_multitime.png']);
22
23 % Error plot
24 figure('Position', [100, 100, 800, 500], 'Color', 'white');
25 semilogy(sim.t, sim.err_L2, 'b-', 'LineWidth', 1.5);
26 xlabel('t'); ylabel('L^2 Error in |u|^2');
27 title(['Case ' num2str(caseID) ': Error Evolution']);
28 grid on;
29 saveas(gcf, [filenames{caseID} '_error.png']);
30
31 % Mass plot
32 figure('Position', [100, 100, 800, 500], 'Color', 'white');
33 plot(sim.t, sim.mass_num, 'b-', 'LineWidth', 1.5); hold on;
34 plot(sim.t, sim.mass_exact, 'r--', 'LineWidth', 1.2);
35 xlabel('t'); ylabel('Mass M(t)');
36 title(['Case ' num2str(caseID) ': Mass Evolution']);
37 legend('Split-Step', 'Exact');
38 grid on;
39 saveas(gcf, [filenames{caseID} '_mass.png']);
40 end

```

Listing 11: Figure generation functions