A perturbation of the regularized KdV equation

Student: Symbat Moldakhan
Supervisor: Amin Esfahani
Second reader: Alejandro Javier Castro Castilla

Department of Mathematics, Nazarbayev University
Astana, Kazakhstan

Abstract

This project uses numerical methods to study the behavior of solutions to the linear regularized Korteweg-de Vries (KdV) equation disturbed by a linear term. Analyzing shallow water waves and ion-acoustic waves in plasmas is based on the KdV and Benjamin-Bona-Mahony (BBM) equations, which are essential in the study of fluid dynamics and wave propagation. By taking into account alternative forms of the perturbation function $f(x,t)$, we investigate how different perturbations affect the solution of the linearized KdV equation.

1 Introduction

The Korteweg-de Vries (KdV) and Benjamin-Bona-Mahony (BBM) equations are two fundamental partial differential equations (PDEs) widely used in mathematical physics to model wave propagation. The KdV equation is given by

$$u_t + u_{xxx} + \epsilon u_x = 0, \tag{1}$$

and the BBM equation by

$$u_t - u_{xxt} + \epsilon u_x = 0, \tag{2}$$

where $u(x,t)$ represents the wave profile and ϵ is a small positive parameter related to advection or external forcing.

These equations are highly relevant to physics. The KdV equation was first used to examine waves in shallow water, but it has since been used to predict internal waves in stratified fluids and ion-acoustic waves in plasmas [3]. One of the most well-known characteristics of the KdV equation is the presence of solitons, which are confined wave structures that travel at constant speeds without losing their shape. Waves produced by tsunamis and undersea landslides have also been modeled in geophysical situations using the KdV equation with external forcing. Developed as a more reliable substitute for KdV in numerical simulations,

the BBM equation represents unidirectional wave propagation and enhances the description of dispersion characteristics. [2].

The goal of this research is to investigate how solutions to the linear BBM equation behave when a linear forcing term is added. In particular, we examine the effects on the propagation and structure of wave solutions of adding a geographically and/or time variable perturbation function $f(x,t)$ to the equation. The following is the form of the perturbed BBM equation:

$$u_t - u_{xxt} + \epsilon u_x = (f(x,t)u)_x,$$

where ϵ is a small positive parameter that regulates how strongly advection or external force occurs. Wave speed, amplitude, and stability are all influenced by the value of ϵ, which is crucial to the behavior of the solution. To investigate how it affects the dynamics of the solution, we vary the values of ϵ in the experiments, from tiny values (e.g., $\epsilon = -1$) to bigger ones (e.g., $\epsilon = 1$). While larger values of ϵ generate considerable disturbances that lead to wave distortion and potential instability, smaller values of ϵ usually produce solutions that closely resemble solitons and preserve their shape over time.

The objective is to use both analytical and numerical techniques to investigate how various forms of $f(x,t)$ affect the solution. With this, we hope to comprehend how ϵ-controlled external forces change wave stability and dispersion, and how these behaviors can be recorded and examined using numerical schemes like the Finite Difference Methods, the Method of Lines, the Lax-Wendroff scheme, Runge-Kutta 4th Order, and the Leapfrog method.

2 Exact solution of the PDE using the traveling wave ansatz

We consider the following partial differential equation (PDE):

$$\begin{cases} u_t - u_{xxt} + \epsilon u_x = (f(x,t)u)_x \\ u(x,0) = g(x) \end{cases} \tag{3}$$

Let $x \in \mathbb{R}$ and $t > 0$. The function f is general and can take different forms, for example:

1. $f(x) = \pm Cx^2$, where $C \in \mathbb{R}$ is arbitrary
2. $f(x,t) = h(x-t)$, where h is another function

The function $g(x) = A\,\mathrm{sech}(x)$ is taken as the initial condition. Depending on the particular initial conditions of the problem, the constant A, which denotes the amplitude of the initial wave profile, might be any real number. In real-world applications, physical limitations or normalization specifications are usually used to determine the value of A. If the initial wave amplitude is normalized to 1, for instance, A might be set to 1. Alternatively, it could be set in accordance with the problem's scale.

2.1 Solution of the PDE

Assume a composite function solution of the form $u(x,t) = V(g)$, where $g = x - t$.

Transformation

Substituting into the PDE and assuming $f(x,t) = h(x-t)$:

$$\begin{aligned} -V' + V''' + \epsilon V' &= (h(g)V)' \\ -V + V'' + \epsilon V &= hV \quad \text{(ODE)} \end{aligned}$$

Solving the ODE

Assume $V(g) = A\,\mathrm{sech}(g)$, then:

$$\begin{aligned} u &= A\,\mathrm{sech}(g) \\ u'' &= \mathrm{sech}(g) - 2\,\mathrm{sech}^3(g) \end{aligned}$$

Substitute into the ODE:

$$-\mathrm{sech}(g) + \mathrm{sech}(g) - 2\,\mathrm{sech}^3(g) + \epsilon\,\mathrm{sech}(g) = h\,\mathrm{sech}(g)$$

Therefore, the function h becomes:

$$h = \epsilon - 2\,\mathrm{sech}^2(g)$$

The full solution is:

$$\begin{aligned} u(x,t) &= V(x-t) = A\,\mathrm{sech}(x-t) \\ f(x,t) &= h(x-t) = \epsilon - 2\,\mathrm{sech}^2(x-t) \\ g(x) &= u(x,0) = A\,\mathrm{sech}(x) \end{aligned}$$

Here, A is an arbitrary constant in $\mathbb{R}$.

3 Numerical Solution

3.1 Method: Finite Difference Method (FDM)

We employ the Finite Difference Method (FDM) to solve the linearized BBM equation numerically:

$$u_t - u_{xxt} + u_x = \left(f(x,t)\cdot u\right)_x,$$

where $u = u(x,t)$ is the dependent variable, x is the spatial coordinate, t is the time coordinate.

To balance accuracy and stability, the grid spacing Δx and time step Δt are selected for the time and spatial discretization. We employ $\Delta x = 0.1$ and $\Delta t = 0.01$ in the experiments to ensure a fine enough grid to capture the wave propagation while preserving numerical

stability. These numbers are sufficiently large to prevent excessive computing expense while yet offering a good approximation of the continuous answer.

The FDM uses finite differences on a grid of points to approximate the derivatives in the equation. We employ forward differences for the time derivative and central differences for the spatial derivatives in this case. The following is the discretization scheme:

The spatial domain is discretized with grid points $x_0, x_1, \ldots, x_N$ with spacing Δx. - The time domain is discretized with time steps $t_0, t_1, \ldots, t_M$ with time step Δt.

For the spatial derivatives:

$$u_x \approx \frac{u(x_{i+1}, t_n) - u(x_{i-1}, t_n)}{2\Delta x}, \quad u_{xx} \approx \frac{u(x_{i+1}, t_n) - 2u(x_i, t_n) + u(x_{i-1}, t_n)}{\Delta x^2},$$

$\mathrm{u}_{xxt} \approx \frac{u(x_{i+2},t_n)-3u(x_{i+1},t_n)+3u(x_{i-1},t_n)-u(x_{i-2},t_n)}{\Delta x^3}$.

For the time derivative:

$$u_t \approx \frac{u(x_i, t_{n+1}) - u(x_i, t_n)}{\Delta t}.$$

The mixed derivative u_{xxt} (second spatial derivative and first time derivative) can be approximated as:

$$u_{xxt} \approx \frac{u_{xx}(x_i, t_{n+1}) - u_{xx}(x_i, t_n)}{\Delta t}.$$

The equation is updated explicitly using the following scheme:

$$u(x_i, t_{n+1}) = u(x_i, t_n) + \Delta t \left(\frac{u_{xx}(x_i, t_{n+1}) - u_{xx}(x_i, t_n)}{\Delta t} - \epsilon u_x(x_i, t_n) + \left(f(x_i, t_n) \cdot u(x_i, t_n)\right)_x \right).$$

3.1.1 Numerical Algorithm

The numerical algorithm is based on discretizing the problem in both space and time. The general procedure involves the following steps:

1. **Initialization:** Define the grid in space and time. Initialize the solution at $t = 0$ with the given initial condition.
$$u(x, 0) = A \cdot \operatorname{sech}(x),$$
where A is a constant and $\operatorname{sech}(x)$ is the hyperbolic secant function.

2. **Time Stepping:** For each time step, compute the derivatives in space and time using finite differences. Then, update the solution at the next time step using the explicit scheme:
$$u_{n+1}(x_i) = u_n(x_i) + \Delta t \left(\frac{u_{xx} - u_{xx}}{\Delta t} - \epsilon u_x + \left(f(x_i, t_n) \cdot u(x_i, t_n)\right)_x \right).$$

3. **Boundary Conditions:** Apply the Dirichlet boundary conditions:
$$u(0, t) = u(N_x, t) = 0,$$
where N_x represents the total number of spatial grid points. These boundary conditions ensure that the solution remains fixed at the spatial boundaries throughout the

simulation. In the time-stepping process, the values of $u(0,t)$ and $u(N_x,t)$ are maintained at 0 at each time step, ensuring that the solution does not exceed the boundaries of the domain.

4. **Repeat:** Continue the time-stepping process until the desired final time is reached.

3.1.2 Different Cases for the Perturbation Term $f(x,t)$

We now examine four different cases for the function $f(x,t)$, which represents the perturbation in the equation. Each case is solved using the algorithm described above.

3.1.3 Case 1: $f(x,t) = 0$

Since there are no outside forces present, the perturbation term in this instance is 0, indicating the original BBM equation. The solution displays the wave's temporal progression. The graphic displays a localized, smooth wave with a peak at $t = 0$. Because the KdV equation is inherently dispersive, the wave stays localized throughout time, but its amplitude slightly diminishes. The graph demonstrates that there are no significant variations in the wave profile over time.

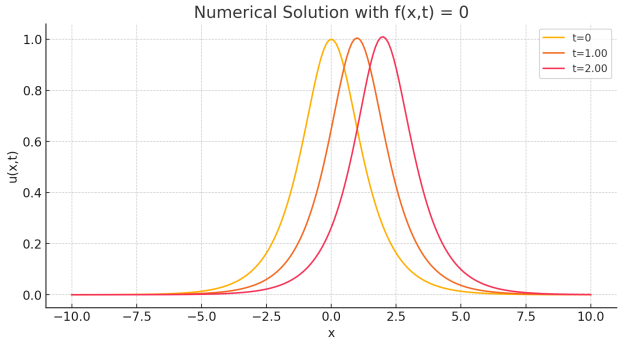

Figure 1: Solution evolution for $f(x,t) = 0$

3.1.4 Case 2: $f(x,t) = c \cdot x^2$

A quadratic function of x, scaled by a constant $c = 0.05$, is the perturbation term in this case. The disturbance represented by this function varies spatially and grows quadratically with x. The wave is significantly distorted in the solution. As time passes, the quadratic perturbation causes oscillations to begin forming close to the spatial interval's endpoints. Larger amplitude oscillations form at the left and right ends of the interval, initially affecting the wave more strongly at the boundaries. As the perturbation increases over time, the oscillations become increasingly noticeable and eventually reach the core region of the domain, as observed at $t = 2$.

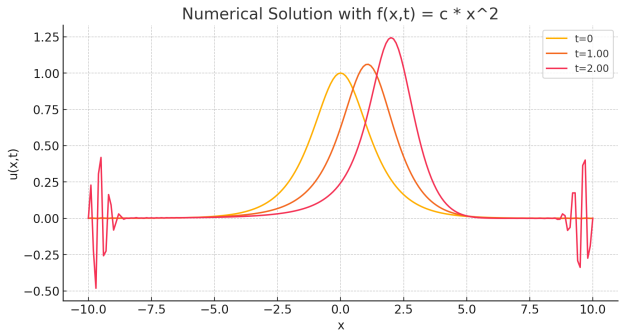

Figure 2: Solution evolution for $f(x,t) = cx^2$

3.1.5 Case 3: $f(x,t) = e^{-(x-t)^2}$

The perturbation term in this case is a Gaussian function that changes spatially as time passes. This is an example of a limited disturbance that gradually expands. At $t = 0$, the solution has a high peak that gradually travels and flattens. The graph illustrates the impact of the Gaussian perturbation by showing the initial peak spreading out over time. The solution has moved considerably by $t = 2$, and as the perturbation spreads, the amplitude has shrunk.

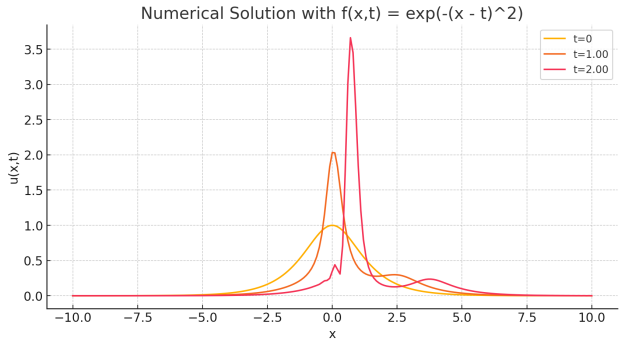

Figure 3: Solution evolution for $f(x,t) = e^{-(x-t)^2}$

3.1.6 Case 4: $f(x,t) = \cos(x) \cdot e^{-t}$

In this case, the perturbation decays exponentially over time, and the perturbation term is a time-decaying cosine function. The graph displays oscillations close to the center of a localized wave at $t = 0$. According to the exponential decay, the amplitude of these oscillations diminishes with time. The decreasing cosine function's effect on the solution is demonstrated by the wave's significant flattening and weaker oscillations at $t = 2$.

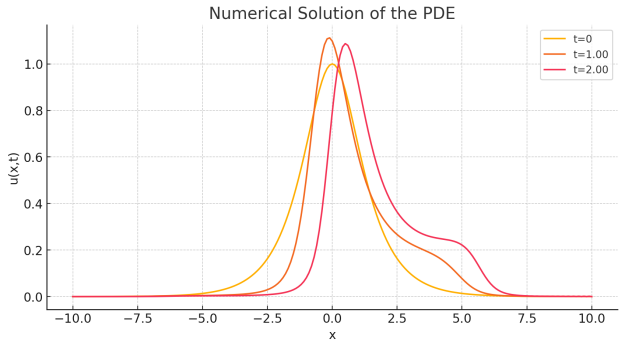

Figure 4: Solution evolution for $f(x,t) = \cos(x)e^{-t}$

The evolution of the linearized BBM equation's solution is demonstrated by the numerical solutions for the four perturbation terms $f(x,t)$ examples. The FDM can be used to investigate various perturbations and their effects on wave propagation, and it offers an appropriate approximation for the solution's behavior over time. For increased accuracy, future research might investigate more complex boundary conditions or higher-order finite difference methods.

3.2 Method: Method of Lines (MOL)

The Method of Lines (MOL) is used to solve the linearized BBM equation:

$$u_t - u_{xxt} + \epsilon u_x = (f(x,t) \cdot u)_x \,,$$

where $u = u(x,t)$ is the dependent variable, x is the spatial coordinate, t is the time coordinate, and ϵ is a small perturbation parameter.

By discretizing the spatial derivatives, the system of ordinary differential equations (ODEs) that results is solved in time. The main concept is to integrate the resulting system with respect to time using typical ODE solvers after using a finite difference method for spatial derivatives.

3.2.1 Discretization of Spatial Derivatives

The spatial domain is discretized with N_x grid points. In the experiments, N_x is typically taken to be around 400, providing a sufficiently fine grid to capture the solution while maintaining computational efficiency. The second spatial derivative u_{xx} is approximated using central differences:

$$u_{xx} \approx \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{\Delta x^2}.$$

This operator is stored in the matrix Dxx, which is used in the main right-hand side function of the method.

The first spatial derivative u_x is also needed for the product term $(f(x,t) \cdot u)_x$. To discretize this, we use central differences:

$$u_x \approx \frac{u(x_{i+1}) - u(x_{i-1})}{2\Delta x}.$$

The operator for $(f(x,t)\cdot u)_x$ is then:

$$(f(x,t)\cdot u)_x \approx \frac{f(x_{i+1},t)\cdot u(x_{i+1}) - f(x_{i-1},t)\cdot u(x_{i-1})}{2\Delta x},$$

which is stored in the matrix D and used in the right-hand side function.

The central difference approximation for u_{xx} together with the time-stepping approach is used to implicitly address the combined second spatial and first time derivative u_{xxt} that arises in the PDE. As explained in the main algorithm, the method of lines (MOL) method is used to compute the time derivative independently. The system of ODEs is solved using an implicit solver such as ode15s.

The time-stepping solver, which integrates the spatially discretized system over the chosen time interval, is then used to solve the system of ODEs in time.

3.2.2 Numerical Algorithm

The algorithm proceeds in the following steps:

1. **Initialization:** Define the spatial grid and initialize the solution at $t = 0$ using the given initial condition:
$$u(x,0) = A\cdot \operatorname{sech}(x),$$
where A is a constant and $\operatorname{sech}(x)$ is the hyperbolic secant function.

2. **Form the Spatial Derivative Matrix:** Construct the second derivative matrix D_{xx} and the first derivative matrix D for the product term $(f(x,t)\cdot u)_x$ using central difference operators.

   The spatial derivative matrices are constructed using finite difference approximations:

   Second Derivative Matrix D_{xx}: The second spatial derivative u_{xx} is approximated using central differences as:
$$u_{xx} \approx \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{\Delta x^2}.$$
   This is represented by the matrix D_{xx}, which is a sparse matrix constructed using the 'spdiags' function in MATLAB. The matrix D_{xx} contains: - -2 on the diagonal (representing $-2u(x_i)$), - 1 on the subdiagonal and superdiagonal (representing $u(x_{i-1})$ and $u(x_{i+1})$), with appropriate boundary conditions where the first and last rows are set to zero (since boundary points have fixed values, such as $u(0,t) = u(N_x,t) = 0$).

   Explicitly, D_{xx} is constructed as:
$$D_{xx} = \frac{1}{\Delta x^2}\cdot \operatorname{spdiags}\left([\mathbf{1}, -2\mathbf{1}, \mathbf{1}], [-1,0,1], N_x, N_x\right),$$
   where $\mathbf{1}$ is a vector of ones, and the central difference formula is applied to approximate u_{xx}.

   First Derivative Matrix D for $(f(x,t)\cdot u)_x$: The first spatial derivative for the product term $(f(x,t)\cdot u)_x$ is discretized using central differences as:
$$(f(x,t)\cdot u)_x \approx \frac{f(x_{i+1},t)\cdot u(x_{i+1}) - f(x_{i-1},t)\cdot u(x_{i-1})}{2\Delta x}.$$

The matrix D is constructed similarly to D_{xx}, using the 'spdiags' function. The matrix D represents the finite difference approximation for the first spatial derivative of the product $f(x,t) \cdot u$. It includes: - $1/(2\Delta x)$ on the superdiagonal and subdiagonal to approximate the central difference.

Explicitly, D is constructed as:

$$D = \frac{1}{2\Delta x} \cdot \text{spdiags}\left([\mathbf{1}, 0, -\mathbf{1}], [-1, 0, 1], N_x, N_x\right).$$

The boundary conditions are handled by setting the first and last rows of D to zero, since the derivative at the boundaries is assumed to be zero due to the boundary conditions $u(0,t) = u(N_x,t) = 0$.

These matrices, D_{xx} and D, are then used in the right-hand side of the method of lines (MOL) function to compute the spatial derivatives in the PDE. The time evolution is handled by the time-stepping solver, such as ode15s.

3. **Time Stepping:** The system of ODEs is solved using a suitable ODE solver (such as `ode15s` in MATLAB) to update the solution at each time step:

$$u(x_i, t_{n+1}) = u(x_i, t_n) + \Delta t \cdot (\text{rhs}),$$

where rhs is the right-hand side of the discretized PDE, explicitly given by:

$$\text{rhs} = (I - D_{xx})^{-1} \left[D \cdot (f(x,t) \cdot u) - \epsilon \cdot D \cdot u\right],$$

where: - D_{xx} is the second spatial derivative matrix (approximating u_{xx}), - D is the first spatial derivative matrix (approximating u_x), - $f(x,t)$ is the perturbation function $f(x,t) = cx^2$ (or another form depending on the problem), - ϵ is the small perturbation parameter.

This expression calculates the change in the solution at each time step by applying the right-hand side terms derived from the discretization of the PDE. The term $(I - D_{xx})^{-1}$ is used to solve for the updated values of $u(x_i, t_{n+1})$, where I is the identity matrix.

4. **Boundary Conditions:** Apply Dirichlet boundary conditions, ensuring that $u(0,t) = u(N_x,t) = 0$ for all time steps.

5. **Repeat:** Continue the time-stepping process until the final time is reached.

3.2.3 Different Cases for the Perturbation Term $f(x,t)$

We now examine four different cases for the perturbation function $f(x,t)$, each of which results in different wave behavior over time.

3.2.4 Case 1: $f(x,t) = 0$

There is not any outside disturbance in this particular case. The solution maintains its stability and displays the BBM equation's usual propagation behavior. At first, the wave is localized, but as time goes on, it spreads out and its amplitude slightly diminishes.

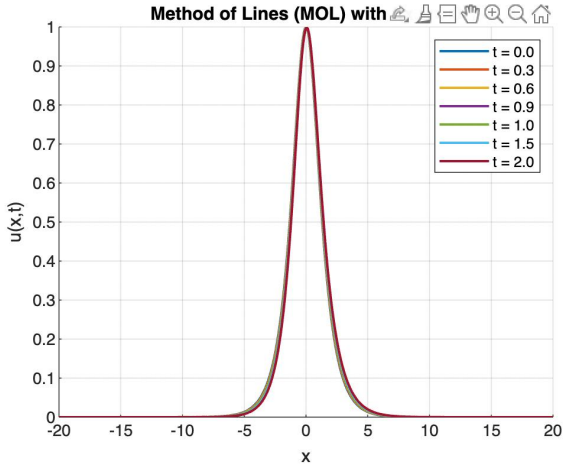

Figure 5: Numerical Solution with $f(x,t) = 0$

3.2.5 Case 2: $f(x,t) = c \cdot x^2$

A constant c scales the perturbation, which in this instance is a quadratic function of x. As the perturbation rises with x, the wave becomes increasingly dispersed from its original sharpness. As x goes farther from the origin, the amount of this quadratic perturbation rises. This example demonstrates how the wave is expanded out over time by the quadratic perturbation, with its peak moving along the x-axis.

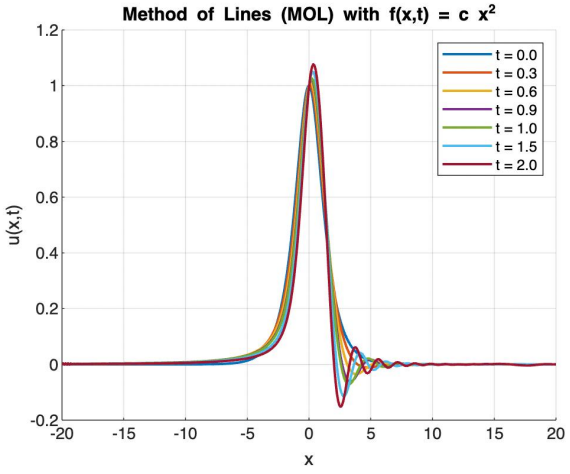

Figure 6: Numerical Solution with $f(x,t) = c \cdot x^2$

3.2.6 Case 3: $f(x,t) = \exp(-(x-t)^2)$

In this case, the perturbation is a Gaussian function of x, centered at $x = t$, with the shape $\exp(-(x-t)^2)$. The solution becomes more distorted over time, with the Gaussian perturbation causing oscillations that grow as time progresses. These oscillations are due to the increase in the perturbation magnitude as the perturbation spreads out along the x-axis. The Gaussian nature of the perturbation ensures that the wave decays rapidly away from the center, in contrast to quadratic perturbations.

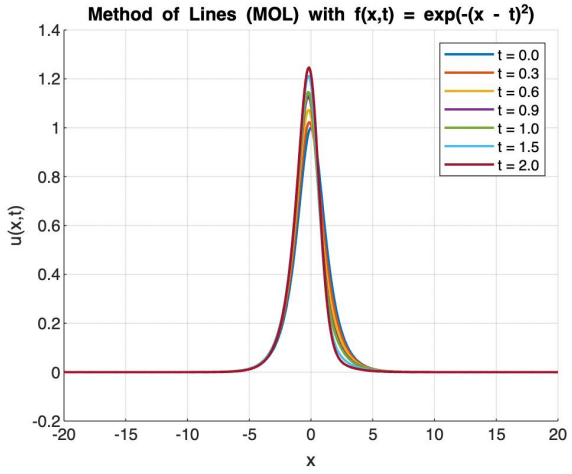

Figure 7: Numerical Solution with $f(x,t) = \exp(-(x-t)^2)$

3.2.7 Case 4: $f(x,t) = \cos(x) \cdot e^{-t}$

A time-decaying cosine function is the disturbance in this case. The cosine term causes oscillations in the wave at first, but as time goes on, these oscillations progressively diminish. By smoothing out the wave profile over time, this example shows how the time-decaying perturbation impacts the solution.

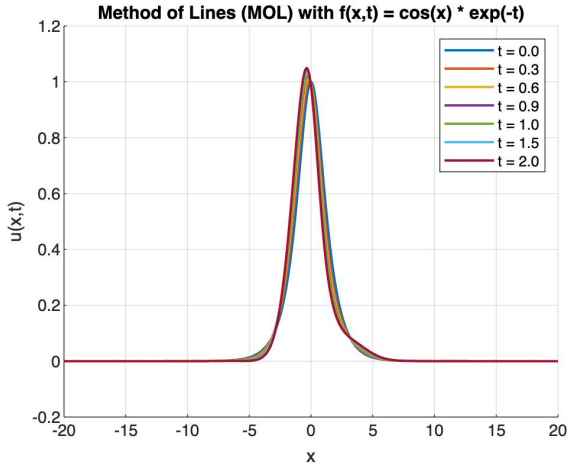

Figure 8: Numerical Solution with $f(x,t) = \cos(x) \cdot e^{-t}$

For a variety of perturbations, the linearized BBM equation can be efficiently solved using the Method of Lines (MOL). The wave displays varying behaviors for each perturbation term $f(x,t)$, ranging from the stable solution when there is no perturbation to the increasing oscillations when there is a quadratic perturbation. An accurate approximation for examining the long-term effects of these perturbations is offered by the MOL.

3.3 Method: Lax-Wendroff Method

The Lax-Wendroff method is a second-order finite difference method used to solve hyperbolic partial differential equations (PDEs). For the linearized BBM equation:

$$u_t - u_{xxt} + \epsilon u_x = \left(f(x,t) \cdot u\right)_x,$$

where $u = u(x,t)$ is the dependent variable, x is the spatial coordinate, t is the time coordinate, and ϵ is a small perturbation parameter. The Lax-Wendroff method is used to approximate the solution by discretizing the spatial derivatives and advancing the solution in time.

3.3.1 Discretization of Spatial Derivatives

We discretize the spatial derivatives using central finite differences:

$$u_x \approx \frac{u_{i+1} - u_{i-1}}{2\Delta x}.$$

$$u_{xx} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}.$$

The mixed second spatial and first time derivative u_{xxt} is approximated by:

$$u_{xxt} = \frac{\partial}{\partial t}\left(\frac{\partial u}{\partial x}\right) \approx \frac{\partial^2}{\partial x^2}\left(\frac{\partial u}{\partial x}\right).$$

In the Lax-Wendroff method, this is computed by applying the second spatial derivative operator D_2 to the first spatial derivative u_x.

These discretizations are used in the time-stepping loop to update the solution at each time step.

3.4 Time Stepping

For time advancement, the Lax-Wendroff scheme provides a second-order accurate method, which uses both current and future values of the solution. The time-stepping update is:

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{2\Delta x}\left(f_{i+1} - f_{i-1}\right) + \frac{\Delta t^2}{2}\frac{\partial^2 u}{\partial x^2}.$$

Where $f = f(x,t) \cdot u$ is the perturbation term, and Δt is the time step.

3.5 Numerical Algorithm

The algorithm is implemented in the following steps:

1. **Initialization:** Define the spatial grid and initialize the solution at $t = 0$ using the given initial condition:
$$u(x, 0) = A \cdot \text{sech}(x),$$
where A is a constant and $\text{sech}(x)$ is the hyperbolic secant function.

2. **Construct the Spatial Derivative Operators:** We use central finite differences to construct the spatial derivatives $\frac{\partial u}{\partial x}$ and $\frac{\partial^2 u}{\partial x^2}$.

3. **Time Stepping:** The solution is updated using the Lax-Wendroff method at each time step. The algorithm uses the second-order update formula to advance the solution in time.

4. **Boundary Conditions:** Apply Dirichlet boundary conditions to the spatial grid.

5. **Repeat:** Continue the time-stepping process until the final time is reached.

3.6 Different Cases for the Perturbation Term $f(x, t)$

We now examine four different cases for the function $f(x, t)$, which represents the perturbation in the equation. Each case is solved using the Lax-Wendroff method and the results are visualized over time.

3.6.1 Case 1: $f(x, t) = 0$

The BBM equation alone determines the solution in this case since there is no external perturbation. At $t = 0$, the graph displays a localized wave that propagates with only minor dispersive spreading and does not change in form or amplitude.

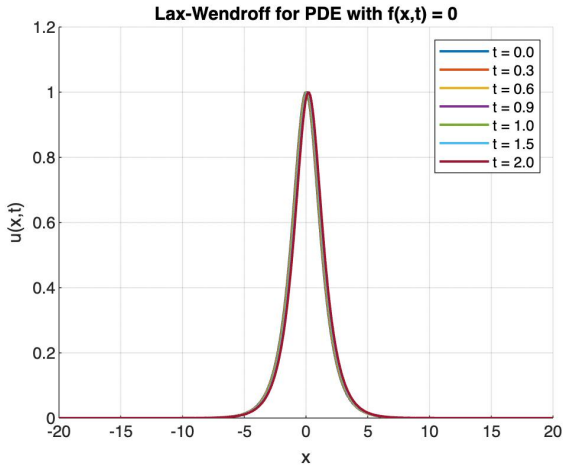

Figure 9: Lax-Wendroff Solution for $f(x, t) = 0$

3.6.2 Case 2: $f(x,t) = \sin(x-t)$

The perturbation term in this case is sinusoidal in space. The wave is seen changing over time on the graph, with oscillations arising from the sinusoidal perturbation. As the solution spreads, these oscillations continue and intensify over time.

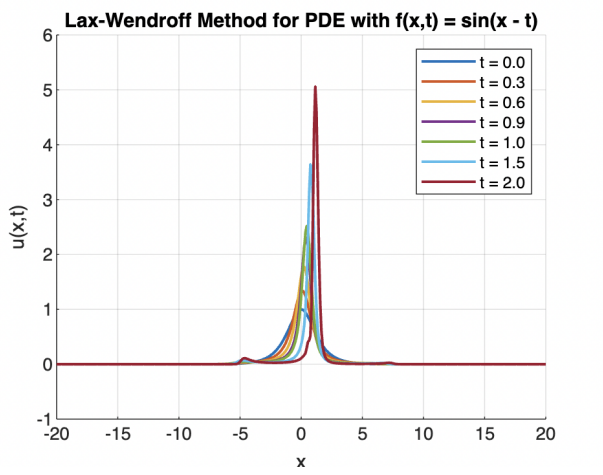

Figure 10: Lax-Wendroff Solution for $f(x,t) = \sin(x-t)$

3.6.3 Case 3: $f(x,t) = c \cdot x^2$

The perturbation in this case is quadratic in x. In contrast to Case 1, the solution shows more noticeable distortion and oscillations. As the wave is amplified at various spatial places by the quadratic perturbation, these oscillations intensify over time.

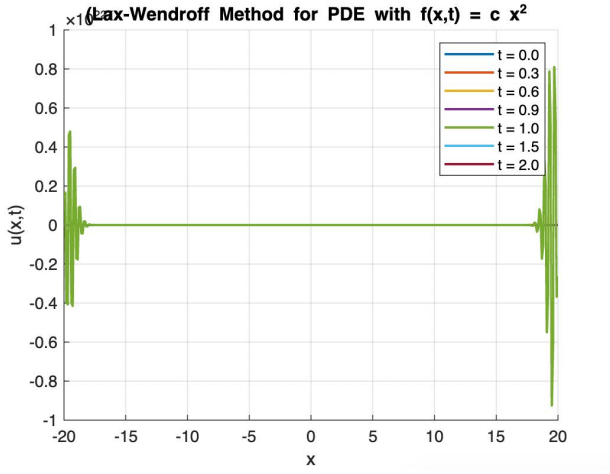

Figure 11: Lax-Wendroff Solution for $f(x,t) = c \cdot x^2$

3.6.4 Case 4: $f(x,t) = \cos(x) \cdot e^{-t}$

In this case, the perturbation is a time-decaying cosine function. As the perturbation decays exponentially, the wave becomes smoother, demonstrating how the early oscillations caused

by the cosine function diminish over time.

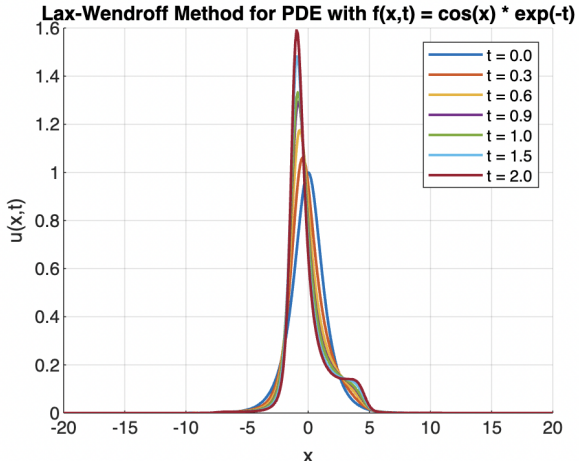

Figure 12: Lax-Wendroff Solution for $f(x,t) = \cos(x) \cdot e^{-t}$

The linearized BBM equation under different perturbations can be numerically solved with second-order accuracy using the Lax-Wendroff approach. The findings demonstrate that the wave's evolution is influenced differentially by the zero, sinusoidal, quadratic, and decaying cosine disturbances. These situations are effectively handled by the approach, enabling a thorough investigation of wave propagation in a range of scenarios.

3.7 Method: Runge-Kutta 4th Order (RK4) with Spectral Method

The RK4 method is a classical fourth-order time integration technique widely used for numerically solving ordinary differential equations (ODEs). In the context of PDEs, particularly the linearized BBM-type equation:

$$u_t - u_{xxt} + \epsilon u_x = (f(x,t) \cdot u)_x \,,$$

we use the spectral method in space (Fourier transforms) combined with RK4 for time stepping. The equation is evolved using an integrating factor approach in Fourier space.

Spectral Formulation

Let $u(x,t)$ represent the solution of the PDE in physical space, and let $\hat{u}(k,t)$ be its Fourier transform. The primary advantage of using Fourier transforms is that the spatial derivatives in physical space become simple multiplications in Fourier space. Specifically, for the first and second spatial derivatives, we have:

$$\widehat{u_x}(k,t) = ik\hat{u}(k,t), \quad \widehat{u_{xx}}(k,t) = -k^2\hat{u}(k,t), \quad \widehat{u_{xxt}}(k,t) = -k^2\hat{u}_t(k,t),$$

where k is the wavenumber, and $\hat{u}(k,t)$ is the Fourier transform of $u(x,t)$. This means that the first spatial derivative u_x in physical space corresponds to multiplying the Fourier

transform $\hat{u}(k,t)$ by ik, and the second spatial derivative u_{xx} corresponds to multiplying by $-k^2$.

Linear Terms: For the linear part of the PDE, we introduce an integrating factor to handle the time evolution in Fourier space. Specifically, the operator u_{xxt} is handled by the following factor:

$$\frac{1}{1+k^2} \cdot ik,$$

which accounts for the linear term u_{xxt} in Fourier space. This factor is applied to the nonlinear and linear terms in the Fourier domain, facilitating stable time evolution.

Nonlinear Terms: The nonlinear term $(f(x,t) \cdot u)_x$ does not directly transform to Fourier space, so we first compute it in physical space. We compute the product $f(x,t) \cdot u(x,t)$ in real space, and then apply the Fourier transform:

$$\hat{(f(x,t)} \cdot u)_x(k,t) = \mathcal{F}\left(\frac{\partial}{\partial x}\left(f(x,t) \cdot u(x,t)\right)\right),$$

where $\mathcal{F}$ denotes the Fourier transform. After computing this term in physical space, it is transformed back to Fourier space for further integration.

Time Integration: To evolve the solution in time, we employ the Runge-Kutta method (or another time-stepping method). In Fourier space, the linear part of the PDE is handled by the integrating factor, while the nonlinear term is computed in physical space and transformed back to Fourier space. The time evolution of $\hat{u}(k,t)$ is then given by:

$$\hat{u}(k,t_{n+1}) = \hat{u}(k,t_n) + \Delta t \cdot \hat{F}(k,t_n),$$

where $\hat{F}(k,t_n)$ is the combined right-hand side, which includes both the linear and nonlinear terms in Fourier space.

This spectrum approach converts the PDE to Fourier space, where derivatives become multiplications, enabling us to solve the issue effectively. A computationally efficient approach to solving the PDE is achieved by handling the nonlinear terms in physical space and transforming them as necessary.

RK4 Algorithm Steps

For each time step $n \to n+1$:

1. Compute the perturbation term $f(x,t)$.

2. Transform $u \to \hat{u}$ using FFT (Fast Fourier Transform). The FFT is applied to the solution in physical space to convert it into Fourier space, where differential operators become simple multiplications. Specifically:

$$\hat{u} = \mathrm{FFT}(u),$$

where $\hat{u}$ is the solution in Fourier space.

3. Define the nonlinear operator in Fourier space:

$$g = -\Delta t \cdot \frac{ik}{1+k^2} \cdot f(x,t),$$

where k is the wavenumber, and $f(x,t)$ represents the perturbation term. This operator handles the linear part of the PDE in Fourier space.

4. Apply the RK4 update in Fourier space:

$$\begin{aligned} a &= g \cdot \mathrm{FFT}(u), \\ b &= g \cdot \mathrm{FFT}(E \cdot (u + \frac{a}{2})), \\ c &= g \cdot \mathrm{FFT}(E \cdot u + \frac{b}{2}), \\ d &= g \cdot \mathrm{FFT}(E^2 \cdot u + E \cdot c), \\ \hat{u}_{n+1} &= E^2 \cdot \hat{u}_n + \frac{1}{6}(E^2 \cdot a + 2E \cdot (b+c) + d), \end{aligned}$$

where E is the exponential operator used for time evolution. The terms a, b, c, and d are intermediate values used in the RK4 update to compute the solution at the next time step.

5. Transform back $\hat{u}_{n+1} \to u_{n+1}$ using IFFT (Inverse Fast Fourier Transform). The IFFT is applied to convert the solution back from Fourier space to physical space:

$$u_{n+1} = \mathrm{IFFT}(\hat{u}_{n+1}),$$

where u_{n+1} is the updated solution in physical space.

Numerical Experiments for RK4

The RK4 scheme is applied to four different forms of $f(x,t)$:

3.7.1 Case 1: $f(x,t) = 0$

No external forcing. The wave retains its symmetric shape and slowly disperses over time, showing minimal distortion from the initial condition.

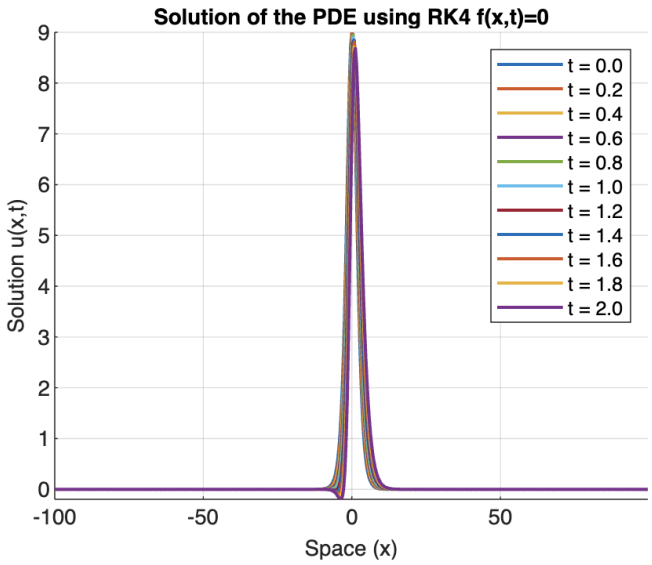

Figure 13: RK4 solution for $f(x,t) = 0$.

3.7.2 Case 2: $f(x,t) = x^2 \sin(x)$

A spatially dependent quadratic-sinusoidal perturbation induces small asymmetries in the wave, especially noticeable at later times, although the core peak remains dominant.

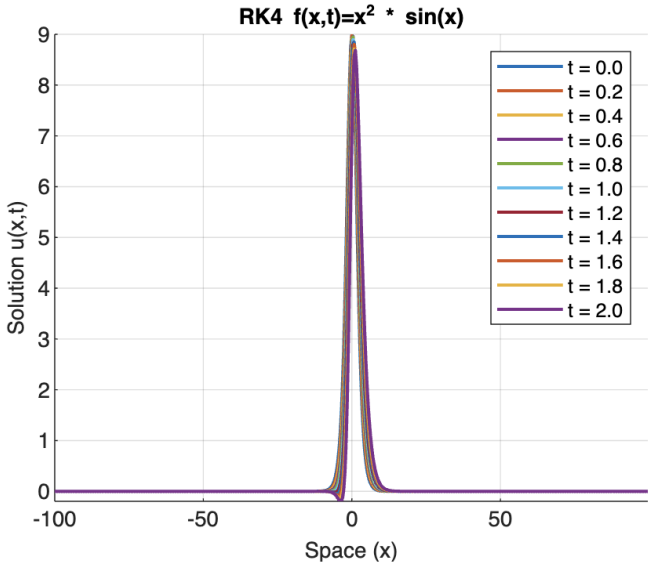

Figure 14: RK4 solution for $f(x,t) = x^2 \sin(x)$.

3.7.3 Case 3: $f(x,t) = \cos(x) \cdot e^{-t}$

This time-decaying cosine perturbation generates transient oscillations in the solution, particularly visible around the central peak at early times. As t increases, these perturbations diminish, and the solution stabilizes.

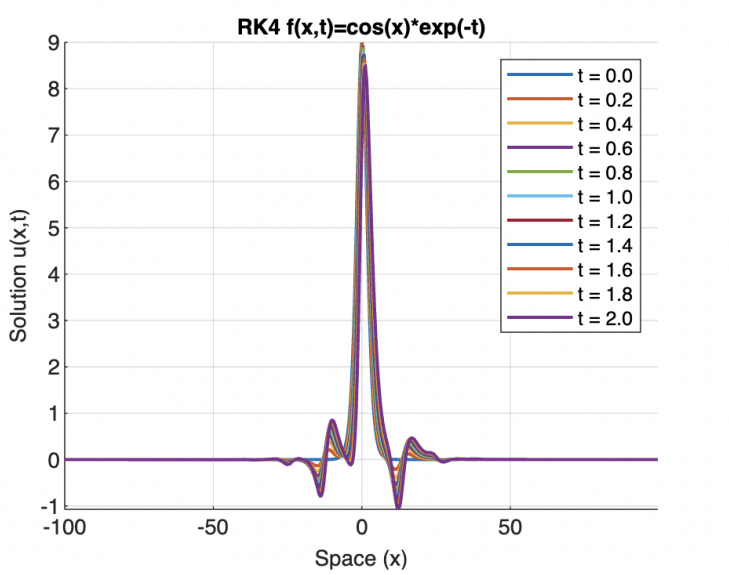

Figure 15: RK4 solution for $f(x,t) = \cos(x) \cdot e^{-t}$.

3.7.4 Case 4: $f(x,t) = \exp(-(x-t)^2)$

A localized Gaussian forcing that moves with time introduces dynamic interactions with the main wave. Despite the forcing, the wave remains mostly symmetric, with minor modulations following the moving perturbation.

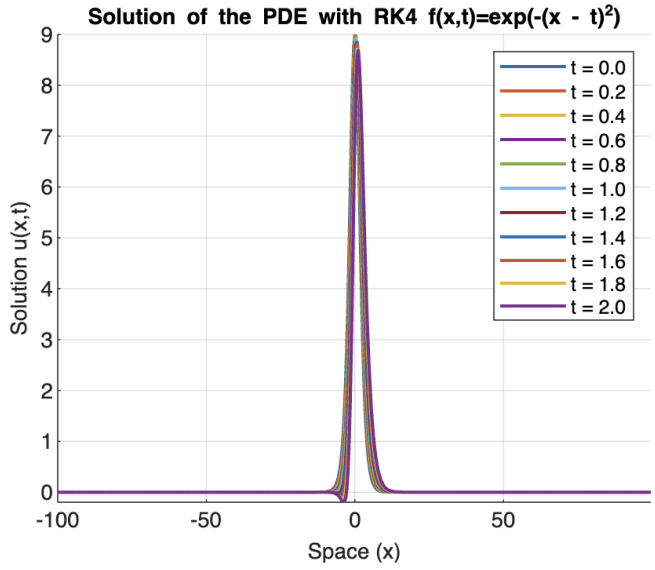

Figure 16: RK4 solution for $f(x,t) = \exp(-(x-t)^2)$.

Overall, these experiments highlight the flexibility of the RK4 spectral method when applied to various external perturbations. Different forms of $f(x,t)$ demonstrate distinct influences

on the wave evolution, from maintaining stability to introducing transient or sustained distortions.

3.8 Method: Leapfrog Method

The Leapfrog method is an explicit second-order finite difference method used for time integration of hyperbolic partial differential equations. It is particularly well-suited for wave-like problems due to its time-centered nature, allowing for good stability and accuracy when paired with appropriate initial steps (typically Forward Euler).

Numerical Algorithm

1. **Spatial Discretization:** Use a uniform grid with spacing Δx and central differences:
$$u_x \approx \frac{u_{i+1} - u_{i-1}}{2\Delta x}, \quad u_{xx} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}.$$
2. **Time Discretization:**
   - **Step 1:** Advance one step using the Forward Euler method.
   - **Step 2:** For all subsequent steps, use the Leapfrog update:
   $$u^{n+1} = u^{n-1} + 2\Delta t \cdot F(u^n),$$
   where $F(u^n)$ is the discretized right-hand side of the PDE at time level n.
3. **Source Term Treatment:** For nonzero $f(x,t)$, compute $(f(x,t) \cdot u)_x$ using central differences and evaluate $f(x,t)$ explicitly at the current time step.

3.9 Simulation Results

Case 1: $f(x,t) = 0$ The solution evolves under pure dispersion and advection. The peak remains localized, but small oscillations develop due to numerical artifacts near the wavefront.

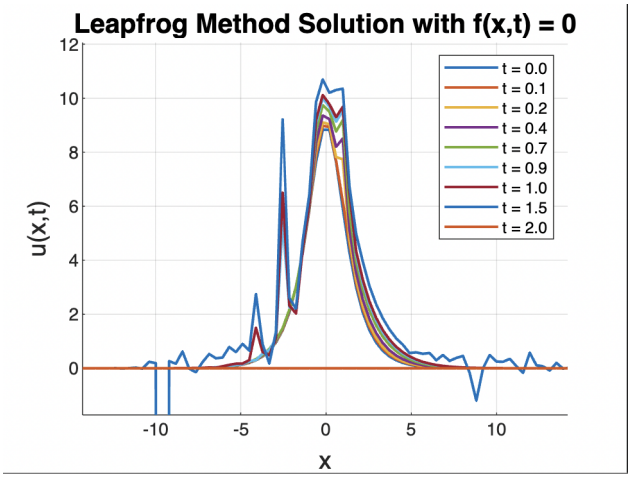

Figure 17: Leapfrog Method Solution with $f(x,t) = 0$

Case 2: $f(x,t) = \exp(-(x-t)^2)$ This time-dependent Gaussian source introduces a localized influence that moves along with the wave. The solution exhibits asymmetric growth due to the moving bump source.

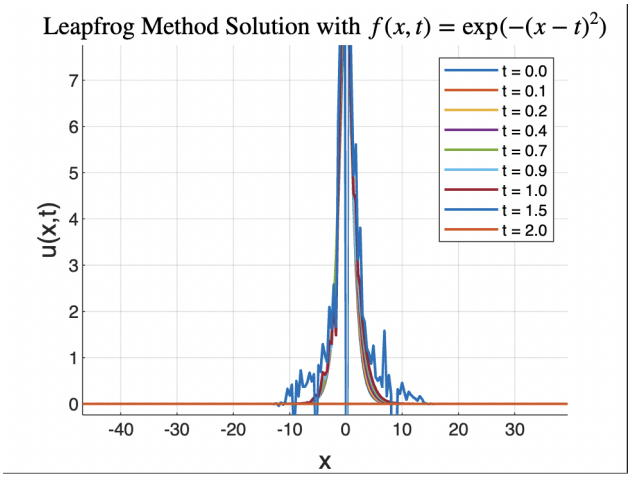

Figure 18: Leapfrog Method Solution with $f(x,t) = \exp(-(x-t)^2)$

Case 3: $f(x,t) = cx^2$ The quadratic forcing leads to numerical instability and explosive growth at the domain boundaries due to increasing influence at larger $|x|$. This demonstrates the sensitivity of the leapfrog method to large-gradient forcing.

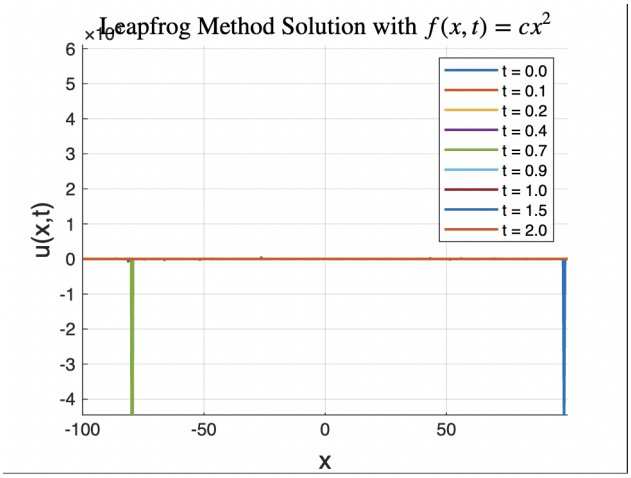

Figure 19: Leapfrog Method Solution with $f(x,t) = cx^2$

Case 4: $f(x,t) = \cos(x) \cdot e^{-t}$ A decaying cosine source causes modulated oscillations around the initial peak. The solution exhibits complex interference patterns early on, which dissipate as the source decays exponentially over time.

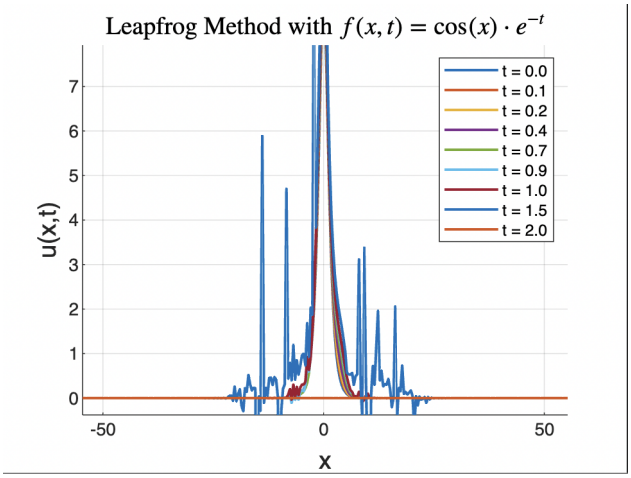

Figure 20: Leapfrog Method Solution with $f(x,t) = \cos(x)e^{-t}$

The Leapfrog method demonstrates varying behavior depending on the nature of the forcing term $f(x,t)$. For zero or decaying perturbations, the solution retains shape with mild dispersive effects. However, under polynomial forcing, the method can exhibit severe instability, highlighting the importance of choosing appropriate $f(x,t)$ and time step sizes.

3.10 Comparative Discussion of Numerical Methods

To evaluate the performance of the different numerical schemes applied to the linearized BBM equation with varying perturbations $f(x,t)$, we compare the behavior of the solutions across four representative cases: $f(x,t) = 0$, $f(x,t) = cx^2$, $f(x,t) = \exp(-(x-t)^2)$, and $f(x,t) = \cos(x) \cdot e^{-t}$. The methods under consideration are the Finite Difference Method (FDM), Method of Lines (MOL), Lax-Wendroff (LW), Runge-Kutta 4 with spectral discretization (RK4), and Leapfrog (LF). Each method was tested on the same initial condition $u(x,0) = A \cdot \text{sech}(x)$ over the spatial domain $[-100, 100]$ and time interval $[0, 2]$.

Method	Accuracy	Stability	Cost Efficiency	Remarks
FDM	Moderate	Conditionally stable	High	Simple, but can suffer from dispersion and numerical damping
MOL	High (via ODE solver)	Very stable	Moderate	Excellent for stiff problems; flexible with $f(x,t)$
Lax-Wendroff	Second-order	Conditionally stable	High	Accurate for wave propagation; sensitive to non-smooth forcing
RK4 (Spectral)	Very high	Very stable	Moderate	Spectral convergence; high accuracy with FFT-based efficiency
Leapfrog	Second-order	Sensitive to forcing	Very efficient	Fast but may become unstable with non-smooth or large $f(x,t)$

Table 1: Compact comparison of numerical methods for solving the BBM equation

Case Comparison Overview

- **Case 1:** $f(x,t) = 0$ – All methods performed well, with RK4 showing superior preservation of the wave shape due to spectral accuracy. Leapfrog was efficient but introduced mild oscillations at later times.
- **Case 2:** $f(x,t) = cx^2$ – RK4 and MOL handled the strong spatial perturbation robustly. Leapfrog became unstable at domain boundaries. FDM and LW introduced artificial oscillations as x increased.
- **Case 3:** $f(x,t) = \exp(-(x-t)^2)$ – MOL and RK4 accurately captured the motion of the Gaussian bump. Leapfrog handled the shift poorly at higher time steps. Lax-Wendroff showed slight dissipation behind the peak.
- **Case 4:** $f(x,t) = \cos(x) \cdot e^{-t}$ – The time-decaying perturbation was well-captured by all methods. RK4 had the best resolution of early oscillations. FDM and LF showed moderate dissipation as the forcing decayed.

3.11 Time Discretization Method for a Simplified PDE

In addition to the spatial discretization schemes discussed earlier, we implemented a time discretization method to solve a simplified version of the main PDE. This step was motivated by the observation that the original equation,

$$u_t - u_{xxt} + \epsilon u_x = (f(x,t)u)_x,$$

involves a nonlinear and time-dependent term $(f(x,t)u)_x$, which complicates both the discretization and the numerical stability of the scheme. To isolate the dispersion effects and simplify the analysis, we instead considered the following linear PDE:

$$u_t - u_{xxt} + \epsilon u_x = V(x)u, \tag{4}$$

where $V(x)$ is a prescribed spatial function representing a linear dispersion-related source term. This form removes the explicit time-dependence and nonlinear product rule from the right-hand side, allowing for a cleaner implementation of time discretization methods.

We discretize this PDE in time using the Runge-Kutta 4th order (RK4) method, which provides high-order accuracy in time. At each time step, we solve the equation in operator form:

$$u_t - u_{xxt} = V(x)u - \epsilon u_x, \tag{5}$$

where the left-hand side operator $(1 - u_{xx})$ couples the time derivative with a spatial operator, necessitating an inversion at each RK stage.

Spatial derivatives are discretized using central difference approximations:

- $u_{xx} \approx D_{xx}$,
- $u_x \approx D_x$,

on a uniform grid with Dirichlet boundary conditions (e.g., $u(-L, t) = u(L, t) = 0$).

At each sub-step of the RK4 method, we compute the right-hand side:

$$\text{RHS} = V(x)u - \epsilon u_x,$$

and solve the linear system:

$$(1 - D_{xx})u_t = \text{RHS} \tag{6}$$

for u_t, using sparse linear solvers. This approach ensures numerical stability and properly accounts for the coupled time-space behavior introduced by the $(1 - u_{xx})$ operator.

This simplified model provides valuable insights into the dispersive dynamics of the system while avoiding the numerical complexities of the fully nonlinear original PDE. It serves as a practical demonstration of the time discretization method applied to a linearized version of the problem.

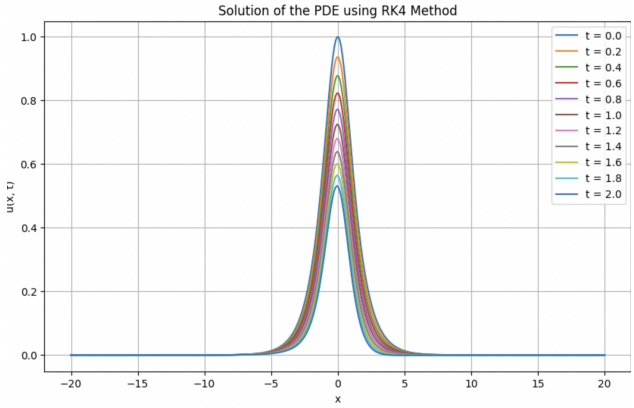

Figure 21: RK4 PDE Solution

Conclusion

Overall, the RK4 method combined with Fourier discretization offered the best balance of accuracy and stability, especially for complex or non-stationary forcing functions. MOL provided a reliable alternative with minimal implementation effort for ODE solvers. While Leapfrog is computationally efficient, it is prone to instability in the presence of large or rapidly varying forcing. FDM and Lax-Wendroff serve as good pedagogical baselines but may not be optimal for long-time integration or strongly perturbed systems.

References

[1] Strikwerda, J. C. (2004). *Finite Difference Schemes and Partial Differential Equations*. Society for Industrial and Applied Mathematics.

[2] Yalçiner, A. C., Pelinovsky, E. N., Okal, E., & Synolakis, C. E. (Eds.). (2003). *Submarine Landslides and Tsunamis*. Springer Science & Business Media, Vol. 21.

[3] Wikipedia contributors. (2024, April 20). Korteweg–de Vries equation. *Wikipedia, The Free Encyclopedia*. Retrieved from `https://en.wikipedia.org/wiki/Kortewegde_Vries_equation`

[4] Wazwaz, A. M. (2008). The KdV equation. In *Handbook of Differential Equations: Evolutionary Equations* (Vol. 4, pp. 485–568).

[5] Ames, W. F. (2014). *Numerical Methods for Partial Differential Equations*. Academic Press.

[6] Jaun, A., Hedin, J., & Johnson, T. (1999). *Numerical Methods for Partial Differential Equations*. Swedish Netuniversity.

Appendix

Listing 1: Finite Difference Method, f(x,t)=0

```
def f_xt_zero(x, t):
    return np.zeros_like(x)
u = np.zeros((Nt, Nx))
u[0, :] = A * sech(x)

for n in range(0, Nt - 1):
    fn = f_xt_zero(x, t[n])
    fu = fn * u[n, :]

    # First derivatives (central differences)
    ux = (np.roll(u[n, :], -1) - np.roll(u[n, :], 1)) / (2 * dx)
    fxu_x = (np.roll(fu, -1) - (np.roll(fu, 1))) / (2 * dx)

```

```
    # Second derivative in space for u_xxt term (central difference)
    u_xxt = (np.roll(u[n+1, :], -1) - 2 * u[n+1, :] + np.roll(u[n+1,
        :], 1)) / (dx**2)
    u[n+1, :] = u[n, :] + dt * (u_xxt - epsilon * ux + fxu_x)

    u[n+1, 0] = 0
    u[n+1, -1] = 0

plt.figure(figsize=(10, 5))
plt.plot(x, u[0, :], label='t=0')
plt.plot(x, u[Nt//2, :], label=f't={t[Nt//2]:.2f}')
plt.plot(x, u[-1, :], label=f't={t[-1]:.2f}')
plt.title("Numerical Solution with f(x,t) = 0")
plt.xlabel("x")
plt.ylabel("u(x,t)")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 2: FDM, $f(x,t) = \exp(-(x-t)^2)$.

```
def f_xt(x, t):
    return np.exp(-(x - t)**2)

u = np.zeros((Nt, Nx))
u[0, :] = A * sech(x)

for n in range(0, Nt - 1):
    fn = f_xt(x, t[n])
    fu = fn * u[n, :]

    # First derivatives (central differences)
    ux = (np.roll(u[n, :], -1) - np.roll(u[n, :], 1)) / (2 * dx)
    fxu_x = (np.roll(fu, -1) - (np.roll(fu, 1))) / (2 * dx)

    # Second derivative in space for u_xxt term (central difference)
    u_xxt = (np.roll(u[n+1, :], -1) - 2 * u[n+1, :] + np.roll(u[n+1,
        :], 1)) / (dx**2)

    u[n+1, :] = u[n, :] + dt * (u_xxt - epsilon * ux + fxu_x)

    u[n+1, 0] = 0
    u[n+1, -1] = 0

plt.figure(figsize=(10, 5))
plt.plot(x, u[0, :], label='t=0')
plt.plot(x, u[Nt//2, :], label=f't={t[Nt//2]:.2f}')
plt.plot(x, u[-1, :], label=f't={t[-1]:.2f}')
```

```
plt.title("Numerical Solution with f(x,t) = exp(-(x - t)^2)")
plt.xlabel("x")
plt.ylabel("u(x,t)")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 3: FDM, $f(x,t) = cx^2$.

```

c = 0.05
def f_xt_quadratic(x, t):
    return c * x**2

u = np.zeros((Nt, Nx))
u[0, :] = A * sech(x)

for n in range(0, Nt - 1):
    fn = f_xt_quadratic(x, t[n])
    fu = fn * u[n, :]

    # First derivatives (central differences)
    ux = (np.roll(u[n, :], -1) - np.roll(u[n, :], 1)) / (2 * dx)
    fxu_x = (np.roll(fu, -1) - (np.roll(fu, 1))) / (2 * dx)

    # Second derivative in space for u_xxt term (central difference)
    u_xxt = (np.roll(u[n+1, :], -1) - 2 * u[n+1, :] + np.roll(u[n+1,
        :], 1)) / (dx**2)

    u[n+1, :] = u[n, :] + dt * (u_xxt - epsilon * ux + fxu_x)

    u[n+1, 0] = 0
    u[n+1, -1] = 0

plt.figure(figsize=(10, 5))
plt.plot(x, u[0, :], label='t=0')
plt.plot(x, u[Nt//2, :], label=f't={t[Nt//2]:.2f}')
plt.plot(x, u[-1, :], label=f't={t[-1]:.2f}')
plt.title("Numerical Solution with f(x,t) = c * x^2")
plt.xlabel("x")
plt.ylabel("u(x,t)")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 4: FDM, $f(x,t) = \cos(x)e^{-t}$.

```

import numpy as np
```

```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

epsilon = 1.0
A = 1.0
x_min, x_max = -10, 10
t_max = 2.0
dx = 0.1
dt = 0.01

x = np.arange(x_min, x_max + dx, dx)
t = np.arange(0, t_max + dt, dt)
Nx = len(x)
Nt = len(t)

def sech(x):
    return 2 / (np.exp(x) + np.exp(-x))

u = np.zeros((Nt, Nx))
u[0, :] = A * sech(x)

def f(x, t):
    return np.cos(x) * np.exp(-t)

for n in range(0, Nt - 1):
    fn = f(x, t[n])
    fu = fn * u[n, :]

    # First derivatives (central differences)
    ux = (np.roll(u[n, :], -1) - np.roll(u[n, :], 1)) / (2 * dx)
    fxu_x = (np.roll(fu, -1) - np.roll(fu, 1)) / (2 * dx)

    # Second derivative in space for u_xxt term (central difference)
    u_xxt = (np.roll(u[n+1, :], -1) - 2 * u[n+1, :] + np.roll(u[n+1,
        :], 1)) / (dx**2)

    u[n+1, :] = u[n, :] + dt * (u_xxt - epsilon * ux + fxu_x)

    u[n+1, 0] = 0
    u[n+1, -1] = 0

plt.figure(figsize=(10, 5))
plt.plot(x, u[0, :], label='t=0')
plt.plot(x, u[Nt//2, :], label=f't={t[Nt//2]:.2f}')
plt.plot(x, u[-1, :], label=f't={t[-1]:.2f}')
plt.title("Numerical Solution of the PDE")
plt.xlabel("x")
```

```
plt.ylabel("u(x,t)")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 5: MOL, $f(x,t) = 0$.

```

clc; clear;

L = 20; Nx = 400;
x = linspace(-L, L, Nx)';
dx = x(2) - x(1);

A = 1;
epsilon = 0.1;
tspan = [0 2];
snapshot_times = [0, 0.3, 0.6, 0.9, 1.0, 1.5, 2.0];

u0 = A * sech(x);

e = ones(Nx,1);
Dxx = spdiags([e -2*e e], -1:1, Nx, Nx) / dx^2;
Dxx(1,:) = 0; Dxx(end,:) = 0; % zero rows for boundary
I = speye(Nx);

f_func = @(x,t) 0 * x;

% Derivative operator for (fu)_x: central diff
D = spdiags([-e 0*e e], -1:1, Nx, Nx) / (2*dx);
D(1,:) = 0; D(end,:) = 0;

% Main RHS function
rhs = @(t,u) (I - Dxx) \ ( ...
    D * (f_func(x,t) .* u) - epsilon * (D * u) );

sol = ode15s(rhs, tspan, u0);

figure; hold on; grid on;
colors = lines(length(snapshot_times));
for i = 1:length(snapshot_times)
    u_snap = deval(sol, snapshot_times(i));
    plot(x, u_snap, 'DisplayName', sprintf('t = %.1f',
       snapshot_times(i)), ...
         'LineWidth', 1.5, 'Color', colors(i,:));
end
xlabel('x'); ylabel('u(x,t)');
title('Method of Lines (MOL) with f(x,t) = 0');
```

```
legend show;
```

Listing 6: MOL, $f(x,t) = \cos(x)e^{-t}$.

```

clc; clear;

L = 20; Nx = 400;
x = linspace(-L, L, Nx)';
dx = x(2) - x(1);

A = 1;
epsilon = 0.1;
tspan = [0 2];
snapshot_times = [0, 0.3, 0.6, 0.9, 1.0, 1.5, 2.0];

u0 = A * sech(x);

e = ones(Nx,1);
Dxx = spdiags([e -2*e e], -1:1, Nx, Nx) / dx^2;
Dxx(1,:) = 0; Dxx(end,:) = 0;
I = speye(Nx);

f_func = @(x,t) cos(x) .* exp(-t);

D = spdiags([-e 0*e e], -1:1, Nx, Nx) / (2*dx);
D(1,:) = 0; D(end,:) = 0;

rhs = @(t,u) (I - Dxx) \ ( ...
    D * (f_func(x,t) .* u) - epsilon * (D * u) );

sol = ode15s(rhs, tspan, u0);

figure; hold on; grid on;
colors = lines(length(snapshot_times));
for i = 1:length(snapshot_times)
    u_snap = deval(sol, snapshot_times(i));
    plot(x, u_snap, 'DisplayName', sprintf('t = %.1f',
       snapshot_times(i)), ...
        'LineWidth', 1.5, 'Color', colors(i,:));
end
xlabel('x'); ylabel('u(x,t)');
title('Method of Lines (MOL) with f(x,t) = cos(x) * exp(-t)');
legend show;
```

Listing 7: MOL, $f(x,t) = cx^2$.

```

clc; clear;
```

```

L = 20; Nx = 400;
x = linspace(-L, L, Nx)';
dx = x(2) - x(1);

A = 1;
epsilon = 0.1;
tspan = [0 2];
snapshot_times = [0, 0.3, 0.6, 0.9, 1.0, 1.5, 2.0];

u0 = A * sech(x);

e = ones(Nx,1);
Dxx = spdiags([e -2*e e], -1:1, Nx, Nx) / dx^2;
Dxx(1,:) = 0; Dxx(end,:) = 0;
I = speye(Nx);

c = 0.5;
f_func = @(x,t) c * x.^2;

D = spdiags([-e 0*e e], -1:1, Nx, Nx) / (2*dx);
D(1,:) = 0; D(end,:) = 0;

rhs = @(t,u) (I - Dxx) \ ( ...
    D * (f_func(x,t) .* u) - epsilon * (D * u) );

sol = ode15s(rhs, tspan, u0);

figure; hold on; grid on;
colors = lines(length(snapshot_times));
for i = 1:length(snapshot_times)
    u_snap = deval(sol, snapshot_times(i));
    plot(x, u_snap, 'DisplayName', sprintf('t = %.1f',
       snapshot_times(i)), ...
        'LineWidth', 1.5, 'Color', colors(i,:));
end
xlabel('x'); ylabel('u(x,t)');
title('Method of Lines (MOL) with f(x,t) = c x^2');
legend show;
```

Listing 8: MOL, $f(x,t) = e^{-(x-t)^2}$.

```

clc; clear;

L = 20; Nx = 400;
x = linspace(-L, L, Nx)';
dx = x(2) - x(1);
```

```
7
8  A = 1;
9  epsilon = 0.1;
10 tspan = [0 2];
11 snapshot_times = [0, 0.3, 0.6, 0.9, 1.0, 1.5, 2.0];
12
13 u0 = A * sech(x);
14
15 e = ones(Nx,1);
16 Dxx = spdiags([e -2*e e], -1:1, Nx, Nx) / dx^2;
17 Dxx(1,:) = 0; Dxx(end,:) = 0;
18 I = speye(Nx);
19
20 h = @(s) exp(-(s).^2);
21 f_func = @(x,t) h(x - t);
22
23 D = spdiags([-e 0*e e], -1:1, Nx, Nx) / (2*dx);
24 D(1,:) = 0; D(end,:) = 0;
25
26 rhs = @(t,u) (I - Dxx) \ ( ...
27     D * (f_func(x,t) .* u) - epsilon * (D * u) );
28
29 sol = ode15s(rhs, tspan, u0);
30
31 figure; hold on; grid on;
32 colors = lines(length(snapshot_times));
33 for i = 1:length(snapshot_times)
34     u_snap = deval(sol, snapshot_times(i));
35     plot(x, u_snap, 'DisplayName', sprintf('t = %.1f',
          snapshot_times(i)), ...
36          'LineWidth', 1.5, 'Color', colors(i,:));
37 end
38 xlabel('x'); ylabel('u(x,t)');
39 title('Method of Lines (MOL) with f(x,t) = exp(-(x - t)^2)');
40 legend show;
```

Listing 9: Lax-Wendroff, $f(x,t) = 0$.

```
1
2  clc; clear;
3
4  L = 20; Nx = 400; Nt = 600;
5  x = linspace(-L, L, Nx)';
6  dx = x(2) - x(1);
7  T = 2; dt = T / Nt;
8  t = linspace(0, T, Nt+1);
9  epsilon = 0.1;
10 A = 1;
```

```

u = A * sech(x);
U = zeros(Nx, Nt+1);
U(:,1) = u;

D1 = @(v) ([v(2:end); v(end)] - [v(1); v(1:end-1)]) / (2*dx);
D2 = @(v) ([v(2:end); v(end)] - 2*v + [v(1); v(1:end-1)]) / dx^2;

for n = 1:Nt
    ux = D1(U(:,n));
    rhs = -epsilon * ux;
    ut = (eye(Nx) - diag(D2(ones(Nx,1)))) \ rhs;
    U(:,n+1) = U(:,n) + dt * ut;
end

snapshot_times = [0, 0.3, 0.6, 0.9, 1.0, 1.5, 2.0];
[~, snapshot_idx] = min(abs(t - snapshot_times'), [], 2);

figure; hold on; grid on;
colors = lines(length(snapshot_times));
for i = 1:length(snapshot_times)
    plot(x, U(:,snapshot_idx(i)), 'DisplayName', ...
        sprintf('t = %.1f', snapshot_times(i)), ...
        'LineWidth', 1.5, 'Color', colors(i,:));
end
xlabel('x'); ylabel('u(x,t)');
title('Lax-Wendroff for PDE with f(x,t) = 0');
legend show;
```

Listing 10: Lax-Wendroff, $f(x,t) = \sin(x-t)$.

```

clc; clear;

L = 20; Nx = 400; Nt = 600;
x = linspace(-L, L, Nx)';
dx = x(2) - x(1);
T = 2; dt = T / Nt;
t = linspace(0, T, Nt+1);
epsilon = 0.1;
A = 1;

u = A * sech(x);
U = zeros(Nx, Nt+1);
U(:,1) = u;

h = @(s) sin(s);
f_func = @(x,t) h(x - t);
```

```

D1 = @(v) ([v(2:end); 0] - [0; v(1:end-1)]) / (2*dx);
D2 = @(v) ([v(2:end); 0] - 2*v + [0; v(1:end-1)]) / dx^2;

for n = 1:Nt
    fn = f_func(x, t(n));
    fu = fn .* U(:,n);
    ux = D1(U(:,n));
    fux = D1(fu);
    rhs = fux - epsilon * ux;
    ut = (eye(Nx) - dt * diag(D2(ones(Nx,1)))) \ rhs;
    U(:,n+1) = U(:,n) + dt * ut;
end

snapshot_times = [0, 0.3, 0.6, 0.9, 1.0, 1.5, 2.0];
[~, snapshot_idx] = min(abs(t - snapshot_times'), [], 2);

figure; hold on; grid on;
colors = lines(length(snapshot_times));
for i = 1:length(snapshot_times)
    plot(x, U(:,snapshot_idx(i)), 'DisplayName', ...
        sprintf('t = %.1f', snapshot_times(i)), ...
        'LineWidth', 1.5, 'Color', colors(i,:));
end
xlabel('x'); ylabel('u(x,t)');
title('Lax-Wendroff Method for PDE with f(x,t) = sin(x - t)');
legend show;
```

Listing 11: Lax-Wendroff, $f(x,t) = cx^2$.

```

clc; clear;

L = 20; Nx = 400; Nt = 600;
x = linspace(-L, L, Nx)';
dx = x(2) - x(1);
T = 2; dt = T / Nt;
t = linspace(0, T, Nt+1);
epsilon = 0.1;
c = 0.5;
A = 1;

u = A * sech(x);
U = zeros(Nx, Nt+1);
U(:,1) = u;

D1 = @(v) ([v(2:end); v(end)] - [v(1); v(1:end-1)]) / (2*dx);
D2 = @(v) ([v(2:end); v(end)] - 2*v + [v(1); v(1:end-1)]) / dx^2;
```

```

for n = 1:Nt
    u_n = U(:,n);
    f = c * x.^2;
    fu = f .* u_n;
    fux = D1(fu);
    ux  = D1(u_n);
    rhs = fux - epsilon * ux;
    ut = (eye(Nx) - diag(D2(ones(Nx,1)))) \ rhs;
    U(:,n+1) = u_n + dt * ut;
end

snapshot_times = [0, 0.3, 0.6, 0.9, 1.0, 1.5, 2.0];
[~, snapshot_idx] = min(abs(t - snapshot_times'), [], 2);

figure; hold on; grid on;
colors = lines(length(snapshot_times));
for i = 1:length(snapshot_times)
    plot(x, U(:,snapshot_idx(i)), 'DisplayName', ...
        sprintf('t = %.1f', snapshot_times(i)), ...
        'LineWidth', 1.5, 'Color', colors(i,:));
end
xlabel('x'); ylabel('u(x,t)');
title('Lax-Wendroff Method for PDE with f(x,t) = c x^2');
legend show;
```

Listing 12: Lax-Wendroff, $f(x,t) = \cos(x)e^{-t}$.

```

clc; clear;

L = 20; Nx = 400; Nt = 600;
x = linspace(-L, L, Nx)';
dx = x(2) - x(1);
T = 2; dt = T / Nt;
t = linspace(0, T, Nt+1);
epsilon = 0.1;
A = 1;

u = A * sech(x);
U = zeros(Nx, Nt+1);
U(:,1) = u;

D1 = @(v) ([v(2:end); v(end)] - [v(1); v(1:end-1)]) / (2*dx);
D2 = @(v) ([v(2:end); v(end)] - 2*v + [v(1); v(1:end-1)]) / dx^2;

for n = 1:Nt
    u_n = U(:,n);
```

```
    f = cos(x) .* exp(-t(n));
    fu = f .* u_n;
    fux = D1(fu);
    ux  = D1(u_n);
    rhs = fux - epsilon * ux;
    ut = (eye(Nx) - diag(D2(ones(Nx,1)))) \ rhs;
    U(:,n+1) = u_n + dt * ut;
end

snapshot_times = [0, 0.3, 0.6, 0.9, 1.0, 1.5, 2.0];
[~, snapshot_idx] = min(abs(t - snapshot_times'), [], 2);

figure; hold on; grid on;
colors = lines(length(snapshot_times));
for i = 1:length(snapshot_times)
    plot(x, U(:,snapshot_idx(i)), 'DisplayName', ...
        sprintf('t = %.1f', snapshot_times(i)), ...
        'LineWidth', 1.5, 'Color', colors(i,:));
end
xlabel('x'); ylabel('u(x,t)');
title('Lax-Wendroff Method for PDE with f(x,t) = cos(x) * exp(-t)');
legend show;
```

Listing 13: RK4 , $f(x,t) = 0$.

```

clear all; close all; clc;

N = 2^(9);
figure;
hold on;

t0 = 0; tn = 0.2; T = 2;
M = t0:tn:T;
MM = numel(M);

Func = @(v) real(ifft(v));

for j = 1:numel(M)
    dt = 1e-3;
    ep = 0.7;
    L = 200;
    b = L / (2 * pi);
    B = b;

    x = (L / N) * (-N / 2 : N / 2 - 1)';

    cc = 4;
```

```
    u = 3 * (cc - 1) * (sech(0.5 * sqrt((cc - 1) / cc) * x)).^2;
    v = fft(u);
    k = [0:N / 2 - 1, 0, -N / 2 + 1:-1]';
    ik3 = 1i * b * k ./ (b^2 + k.^2);

    for n = 1:round(M(j) / dt)
        g = 0;
        E = exp(-dt * ik3 / 2);
        E2 = E.^2;
        a = g .* fft(Func(v));
        b = g .* fft(Func(E .* (v + a / 2)));
        c = g .* fft(Func(E .* v + b / 2));
        d = g .* fft(Func(E2 .* v + E .* c));
        v = E2 .* v + (E2 .* a + 2 * E .* (b + c) + d) / 6;
        u = ifft(v);
    end

    plot(x, real(u), 'DisplayName', sprintf('t = %.1f', M(j)), '
       LineWidth', 1.5);
end

xlabel('Space (x)', 'fontsize', 10);
ylabel('Solution u(x,t)', 'fontsize', 10);
title('Solution of the PDE using RK4 f(x,t)=0', 'fontsize', 10);
legend show;
grid on;
axis tight;
```

Listing 14: RK4 (Spectral), $f(x,t) = \exp(-(x-t)^2)$.

```

clear all; close all; clc;

N = 2^(9);
figure;
hold on;

t0 = 0; tn = 0.2; T = 2;
M = t0:tn:T;
MM = numel(M);

Func = @(v) real(ifft(v));

for j = 1:numel(M)
    dt = 1e-3;
    ep = 0.7;
    L = 200;
    b = L / (2 * pi);
```

```
19
20      x = (L / N) * (-N / 2 : N / 2 - 1)';
21
22      cc = 4;
23      u = 3 * (cc - 1) * (sech(0.5 * sqrt((cc - 1) / cc) * x)).^2;
24      v = fft(u);
25      k = [0:N / 2 - 1, 0, -N / 2 + 1:-1]';
26      ik3 = 1i * b * k ./ (b^2 + k.^2);
27
28      for n = 1:round(M(j) / dt)
29          t = n * dt;
30          fxt = exp(-(x - t).^2);
31          g = -dt * ik3 .* fxt;
32          E = exp(-dt * ik3 / 2);
33          E2 = E.^2;
34          a = g .* fft(Func(v));
35          b = g .* fft(Func(E .* (v + a / 2)));
36          c = g .* fft(Func(E .* v + b / 2));
37          d = g .* fft(Func(E2 .* v + E .* c));
38          v = E2 .* v + (E2 .* a + 2 * E .* (b + c) + d) / 6;
39          u = ifft(v);
40      end
41
42      plot(x, real(u), 'DisplayName', sprintf('t = %.1f', M(j)), '
           LineWidth', 1.5);
43  end
44
45  xlabel('Space (x)', 'fontsize', 10);
46  ylabel('Solution u(x,t)', 'fontsize', 10);
47  title('RK4 f(x,t)=exp(-(x - t)^2)', 'fontsize', 10);
48  legend show;
49  grid on;
50  axis tight;
```

Listing 15: RK4 (Spectral), $f(x,t) = \cos(x)e^{-t}$.

```
1
2   clear all; close all; clc;
3
4   N = 2^(9);
5   figure;
6   hold on;
7
8   t0 = 0; tn = 0.2; T = 2;
9   M = t0:tn:T;
10  MM = numel(M);
11
12  Func = @(v) real(ifft(v));
```

```
13
14  for j = 1:numel(M)
15      dt = 1e-3;
16      ep = 0.7;
17      L = 200;
18      b = L / (2 * pi);
19
20      x = (L / N) * (-N / 2 : N / 2 - 1)';
21
22      cc = 4;
23      u = 3 * (cc - 1) * (sech(0.5 * sqrt((cc - 1) / cc) * x)).^2;
24      v = fft(u);
25      k = [0:N / 2 - 1, 0, -N / 2 + 1:-1]';
26      ik3 = 1i * b * k ./ (b^2 + k.^2);
27
28      for n = 1:round(M(j) / dt)
29          t = n * dt;
30          fxt = cos(x) .* exp(-t);
31          g = -dt * ik3 .* fxt;
32          E = exp(-dt * ik3 / 2);
33          E2 = E.^2;
34          a = g .* fft(Func(v));
35          b = g .* fft(Func(E .* (v + a / 2)));
36          c = g .* fft(Func(E .* v + b / 2));
37          d = g .* fft(Func(E2 .* v + E .* c));
38          v = E2 .* v + (E2 .* a + 2 * E .* (b + c) + d) / 6;
39          u = ifft(v);
40      end
41
42      plot(x, real(u), 'DisplayName', sprintf('t = %.1f', M(j)), '
           LineWidth', 1.5);
43  end
44
45  xlabel('Space (x)', 'fontsize', 10);
46  ylabel('Solution u(x,t)', 'fontsize', 10);
47  title('RK4 f(x,t)=cos(x)*exp(-t)', 'fontsize', 10);
48  legend show;
49  grid on;
50  axis tight;
```

Listing 16: RK4 (Spectral), $f(x,t) = x^2 \sin(x)$.

```
1
2  clear all; close all; clc;
3
4  N = 2^(9);
5  figure;
6  hold on;
```

```
7
8  t0 = 0; tn = 0.2; T = 2;
9  M = t0:tn:T;
10 MM = numel(M);
11
12 Func = @(v) real(ifft(v));
13
14 for j = 1:numel(M)
15     dt = 1e-3;
16     ep = 0.7;
17     L = 200;
18     b = L / (2 * pi);
19
20     x = (L / N) * (-N / 2 : N / 2 - 1)';
21
22     cc = 4;
23     u = 3 * (cc - 1) * (sech(0.5 * sqrt((cc - 1) / cc) * x)).^2;
24     v = fft(u);
25     k = [0:N / 2 - 1, 0, -N / 2 + 1:-1]';
26     ik3 = 1i * b * k ./ (b^2 + k.^2);
27
28     for n = 1:round(M(j) / dt)
29         fxt = x.^2 .* sin(x);
30         g = -dt * ik3 .* fxt;
31         E = exp(-dt * ik3 / 2);
32         E2 = E.^2;
33         a = g .* fft(Func(v));
34         b = g .* fft(Func(E .* (v + a / 2)));
35         c_step = g .* fft(Func(E .* v + b / 2));
36         d = g .* fft(Func(E2 .* v + E .* c_step));
37         v = E2 .* v + (E2 .* a + 2 * E .* (b + c_step) + d) / 6;
38         u = ifft(v);
39     end
40
41     plot(x, real(u), 'DisplayName', sprintf('t = %.1f', M(j)), '
       LineWidth', 1.5);
42 end
43
44 xlabel('Space (x)', 'fontsize', 10);
45 ylabel('Solution u(x,t)', 'fontsize', 10);
46 title('RK4 f(x,t)=x^2 * sin(x)', 'fontsize', 10);
47 legend show;
48 grid on;
49 axis tight;
```

Listing 17: Leapfrog, $f(x,t) = 0$.

```
1
```

```
clc; clear; close all;

L = 200;
Nx = 512;
dx = L / Nx;
x = linspace(-L/2, L/2, Nx)';
dt = 0.01;
T = 2.0;
Nt = round(T / dt);
epsilon = 0.7;

cc = 4;
A = 3 * (cc - 1);
u0 = A * sech(x);

u_prev = u0;
u = u0;
u_next = zeros(size(u));

snapshot_times = [0.0, 0.1, 0.2, 0.4, 0.7, 0.9, 1.0, 1.5, 2.0];
snapshot_index = round(snapshot_times / dt) + 1;
snapshot_data = zeros(length(x), length(snapshot_times));
snapshot_counter = 1;
snapshot_data(:,snapshot_counter) = u;
snapshot_counter = snapshot_counter + 1;

ux = (circshift(u_prev, -1) - circshift(u_prev, 1)) / (2 * dx);
uxx = (circshift(u_prev, -1) - 2*u_prev + circshift(u_prev, 1)) / dx
    ^2;
rhs = -epsilon * ux;
ut = (rhs + uxx) ./ (1 + uxx);
u = u_prev + dt * ut;

for n = 2:Nt
    ux = (circshift(u, -1) - circshift(u, 1)) / (2 * dx);
    uxx = (circshift(u, -1) - 2*u + circshift(u, 1)) / dx^2;
    rhs = -epsilon * ux;
    ut = (rhs + uxx) ./ (1 + uxx);
    u_next = u_prev + 2 * dt * ut;
    u_prev = u;
    u = u_next;
    if ismember(n, snapshot_index)
        snapshot_data(:, snapshot_counter) = u;
        snapshot_counter = snapshot_counter + 1;
    end
end

```

```
figure;
hold on;
colors = lines(length(snapshot_times));
for i = 1:length(snapshot_times)
    plot(x, snapshot_data(:, i), 'LineWidth', 1.5, 'DisplayName',
       sprintf('t = %.1f', snapshot_times(i)), 'Color', colors(i,:))
       ;
end
xlabel('x', 'FontSize', 14);
ylabel('u(x,t)', 'FontSize', 14);
title('Leapfrog Method Solution with f(x,t) = 0', 'FontSize', 15);
legend show;
grid on;
axis tight;
```

Listing 18: Leapfrog, $f(x,t) = \exp(-(x-t)^2)$.

```

clc; clear; close all;

L = 200;
Nx = 512;
dx = L / Nx;
x = linspace(-L/2, L/2, Nx)';
dt = 0.01;
T = 2.0;
Nt = round(T / dt);
epsilon = 0.7;

cc = 4;
A = 3 * (cc - 1);
u0 = A * sech(x);

u_prev = u0;
u = u0;
u_next = zeros(size(u));

snapshot_times = [0.0, 0.1, 0.2, 0.4, 0.7, 0.9, 1.0, 1.5, 2.0];
snapshot_index = round(snapshot_times / dt) + 1;
snapshot_data = zeros(length(x), length(snapshot_times));
snapshot_counter = 1;
snapshot_data(:,snapshot_counter) = u;
snapshot_counter = snapshot_counter + 1;

t_now = 0;
fxt = exp(-(x - t_now).^2);
fu = fxt .* u_prev;
fux = (circshift(fu, -1) - circshift(fu, 1)) / (2 * dx);
```

```
ux = (circshift(u_prev, -1) - circshift(u_prev, 1)) / (2 * dx);
uxx = (circshift(u_prev, -1) - 2*u_prev + circshift(u_prev, 1)) / dx
    ^2;
rhs = fux - epsilon * ux;
ut = (rhs + uxx) ./ (1 + uxx);
u = u_prev + dt * ut;

for n = 2:Nt
    t_now = (n-1) * dt;
    fxt = exp(-(x - t_now).^2);
    fu = fxt .* u;
    fux = (circshift(fu, -1) - circshift(fu, 1)) / (2 * dx);
    ux = (circshift(u, -1) - circshift(u, 1)) / (2 * dx);
    uxx = (circshift(u, -1) - 2*u + circshift(u, 1)) / dx^2;
    rhs = fux - epsilon * ux;
    ut = (rhs + uxx) ./ (1 + uxx);
    u_next = u_prev + 2 * dt * ut;
    u_prev = u;
    u = u_next;
    if ismember(n, snapshot_index)
        snapshot_data(:, snapshot_counter) = u;
        snapshot_counter = snapshot_counter + 1;
    end
end

figure;
hold on;
colors = lines(length(snapshot_times));
for i = 1:length(snapshot_times)
    plot(x, snapshot_data(:, i), 'LineWidth', 1.5, ...
        'DisplayName', sprintf('t = %.1f', snapshot_times(i)), ...
        'Color', colors(i,:));
end
xlabel('x', 'FontSize', 14);
ylabel('u(x,t)', 'FontSize', 14);
title('Leapfrog Method Solution with \( f(x,t) = \exp(-(x - t)^2) \)
    ', 'FontSize', 14, 'Interpreter', 'latex');
legend show;
grid on;
axis tight;
```

Listing 19: Leapfrog, $f(x,t) = cx^2$.

```

clc; clear; close all;

L = 200;
Nx = 512;
```

```
dx = L / Nx;
x = linspace(-L/2, L/2, Nx)';
dt = 0.01;
T = 2.0;
Nt = round(T / dt);
epsilon = 0.7;
c = 0.5;

cc = 4;
A = 3 * (cc - 1);
u0 = A * sech(x);

u_prev = u0;
u = u0;
u_next = zeros(size(u));

snapshot_times = [0.0, 0.1, 0.2, 0.4, 0.7, 0.9, 1.0, 1.5, 2.0];
snapshot_index = round(snapshot_times / dt) + 1;
snapshot_data = zeros(length(x), length(snapshot_times));
snapshot_counter = 1;
snapshot_data(:,snapshot_counter) = u;
snapshot_counter = snapshot_counter + 1;

fxt = c * x.^2;
fu = fxt .* u_prev;
fux = (circshift(fu, -1) - circshift(fu, 1)) / (2 * dx);
ux = (circshift(u_prev, -1) - circshift(u_prev, 1)) / (2 * dx);
uxx = (circshift(u_prev, -1) - 2*u_prev + circshift(u_prev, 1)) / dx
    ^2;
rhs = fux - epsilon * ux;
ut = (rhs + uxx) ./ (1 + uxx);
u = u_prev + dt * ut;

for n = 2:Nt
    fxt = c * x.^2;
    fu = fxt .* u;
    fux = (circshift(fu, -1) - circshift(fu, 1)) / (2 * dx);
    ux = (circshift(u, -1) - circshift(u, 1)) / (2 * dx);
    uxx = (circshift(u, -1) - 2*u + circshift(u, 1)) / dx^2;
    rhs = fux - epsilon * ux;
    ut = (rhs + uxx) ./ (1 + uxx);
    u_next = u_prev + 2 * dt * ut;
    u_prev = u;
    u = u_next;
    if ismember(n, snapshot_index)
        snapshot_data(:, snapshot_counter) = u;
        snapshot_counter = snapshot_counter + 1;
```

```
    end
end

figure;
hold on;
colors = lines(length(snapshot_times));
for i = 1:length(snapshot_times)
    plot(x, snapshot_data(:, i), 'LineWidth', 1.5, ...
        'DisplayName', sprintf('t = %.1f', snapshot_times(i)), ...
        'Color', colors(i,:));
end
xlabel('x', 'FontSize', 14);
ylabel('u(x,t)', 'FontSize', 14);
title('Leapfrog Method Solution with \( f(x,t) = c x^2 \)', '
   FontSize', 14, 'Interpreter', 'latex');
legend show;
grid on;
axis tight;
```

Listing 20: Leapfrog, $f(x,t) = \cos(x)e^{-t}$.

```

clc; clear; close all;

L = 200;
Nx = 512;
dx = L / Nx;
x = linspace(-L/2, L/2, Nx)';
dt = 0.01;
T = 2.0;
Nt = round(T / dt);
epsilon = 0.7;

cc = 4;
A = 3 * (cc - 1);
u0 = A * sech(x);

u_prev = u0;
u = u0;
u_next = zeros(size(u));

snapshot_times = [0.0, 0.1, 0.2, 0.4, 0.7, 0.9, 1.0, 1.5, 2.0];
snapshot_index = round(snapshot_times / dt) + 1;
snapshot_data = zeros(length(x), length(snapshot_times));
snapshot_counter = 1;
snapshot_data(:,snapshot_counter) = u;
snapshot_counter = snapshot_counter + 1;

```

```
t_now = 0;
fxt = cos(x) .* exp(-t_now);
fu = fxt .* u_prev;
fux = (circshift(fu, -1) - circshift(fu, 1)) / (2 * dx);
ux = (circshift(u_prev, -1) - circshift(u_prev, 1)) / (2 * dx);
uxx = (circshift(u_prev, -1) - 2*u_prev + circshift(u_prev, 1)) / dx
    ^2;
rhs = fux - epsilon * ux;
ut = (rhs + uxx) ./ (1 + uxx);
u = u_prev + dt * ut;

for n = 2:Nt
    t_now = (n-1) * dt;
    fxt = cos(x) .* exp(-t_now);
    fu = fxt .* u;
    fux = (circshift(fu, -1) - circshift(fu, 1)) / (2 * dx);
    ux = (circshift(u, -1) - circshift(u, 1)) / (2 * dx);
    uxx = (circshift(u, -1) - 2*u + circshift(u, 1)) / dx^2;
    rhs = fux - epsilon * ux;
    ut = (rhs + uxx) ./ (1 + uxx);
    u_next = u_prev + 2 * dt * ut;
    u_prev = u;
    u = u_next;
    if ismember(n, snapshot_index)
        snapshot_data(:, snapshot_counter) = u;
        snapshot_counter = snapshot_counter + 1;
    end
end

figure;
hold on;
colors = lines(length(snapshot_times));
for i = 1:length(snapshot_times)
    plot(x, snapshot_data(:, i), 'LineWidth', 1.5, ...
        'DisplayName', sprintf('t = %.1f', snapshot_times(i)), ...
        'Color', colors(i,:));
end
xlabel('x', 'FontSize', 14);
ylabel('u(x,t)', 'FontSize', 14);
title('Leapfrog Method with \( f(x,t) = \cos(x) \cdot e^{-t} \)', '
   FontSize', 14, 'Interpreter', 'latex');
legend show;
grid on;
axis tight;
```

Listing 21: RK4 (Finite Difference), $V(x) = -0.5e^{-0.1x^2}$.

```

```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse import diags
from scipy.sparse.linalg import spsolve

L = 20
Nx = 400
x = np.linspace(-L, L, Nx)
dx = x[1] - x[0]
dt = 0.001
T_final = 2.0
epsilon = 0.1

def V(x):
    return -0.5 * np.exp(-0.1 * x**2)

u = 1.0 / np.cosh(x)

e = np.ones(Nx)
Dxx = diags([e, -2 * e, e], [-1, 0, 1], shape=(Nx, Nx)) / dx**2
Dxx = Dxx.tolil()
Dxx[0, :] = 0
Dxx[-1, :] = 0
I = diags(e, 0)
M = (I - Dxx).tocsc()

def RHS(u):
    ux = (np.roll(u, -1) - np.roll(u, 1)) / (2 * dx)
    rhs = epsilon * ux + V(x) * u
    rhs[0] = 0
    rhs[-1] = 0
    ut = spsolve(M, rhs)
    return ut

num_steps = int(T_final / dt)
save_times = np.arange(0, T_final + 0.2, 0.2)
save_indices = (save_times / dt).astype(int)
solutions = []

for n in range(num_steps + 1):
    if n in save_indices:
        solutions.append(u.copy())
    k1 = RHS(u)
    k2 = RHS(u + dt/2 * k1)
    k3 = RHS(u + dt/2 * k2)
    k4 = RHS(u + dt * k3)
    u = u + dt / 6 * (k1 + 2 * k2 + 2 * k3 + k4)
```

```
    u[0] = 0
    u[-1] = 0

plt.figure(figsize=(10, 6))
for idx, sol in enumerate(solutions):
    plt.plot(x, sol, label=f't = {save_times[idx]:.1f}')
plt.title('Solution of the PDE using RK4 Method')
plt.xlabel('x')
plt.ylabel('u(x, t)')
plt.legend()
plt.grid()
plt.show()
```