FUDMA Journal of Sciences (FJS)
ISSN online: 2616-1370
ISSN print: 2645 - 2944
Vol. 10 No. 6, March, 2026, pp 302 – 313
DOI: https://doi.org/10.33003/fjs-2026-1006-5171

UNDERSTANDING THE SOLITON PHENOMENA: LEAPFROGGING THROUGH THE KORTEWEG-DE VRIES EQUATION FOR MULTI-SOLITONS SOLUTION

***[1]Paul Shimlumun Amon & [2]Orapine Hycienth Ortser**

[1]Department of Physics, Benue State University, Makurdi, Nigeria.
[2]Department of Mathematics, Faculty of Natural and Applied Sciences, Nigerian Army University Biu, Borno State, Nigeria.

*Corresponding authors' email: pamon@bsum.edu.ng

ABSTRACT

This study applies the Leapfrog algorithm to numerically obtain multi-soliton solutions of the Korteweg–de Vries (KdV) equation. The KdV equation is a classical nonlinear partial differential equation that models wave motion in dispersive media and has been widely used to describe shallow water waves, plasmas, fiber optics, Josephson junctions, and electromagnetic wave propagation. Over the years, it has become a fundamental model for understanding how nonlinearity and dispersion interact in physical systems. One of its most remarkable features is the existence of solitons; stable, localized wave packets that travel without changing shape. These waves have attracted strong interest because of their unusual ability to interact with other solitons and still retain their identity. The Leapfrog algorithm is implemented in MATLAB and Python, two common scientific computing languages. This comparison evaluates the performance, accuracy, and efficiency of these implementations. The simulation results focus on solitons, especially their collision and interaction behaviours. The analytical and numerical solutions match, although the numerical results exhibit small fluctuations in soliton profiles after collisions. However, these fluctuations do not significantly affect their core properties. This supports the existence of soliton solutions and confirms that solitons collide without altering their characteristics or identities. The findings from this study deepen our understanding of the KdV equations and their soliton solutions and provide valuable insights into the computational aspects of solving differential equations in Python and MATLAB.

Keywords: Korteweg-de Vries Equation, Leapfrog, Multi-Solitons, Numerical Study

INTRODUCTION

We consider in this study the Korteweg-de Vries (KdV) equation, given by

$$w_t \pm \beta w w_x + \alpha w_{xxx} = 0, \quad \alpha, \beta \in \mathbb{R}, \qquad (1)$$

with initial condition $w(x,0) = f(x)$, where $w(x,t)$ is a function of two variables which represent the amplitude of the wave at position x, and time t, and α, β are arbitrary positive parameters. The KdV equation is a well-known example of a nonlinear partial differential equation in the field of nonlinear wave phenomena. It has long attracted significant interest from physicists and mathematicians. This equation, derived from the study of shallow-water waves, has been applied across a wide range of disciplines, including plasma physics, fiber optics, Josephson junctions, and electromagnetic waves (Helfrich & Melville, 2006; Haruna et al., 2018).

One of the fascinating features of the KdV equation is its capacity to generate solitons. These solitary wave packets possess remarkable properties: they maintain their shape as they propagate at a constant speed. They can interact with one another without altering their fundamental properties, as illustrated in Figure 1 (Zabusky & Kruskal, 1965). According to Ya (2013), Solitons arise as a result of the balance between the dispersive term (αw_{xxx}) and the nonlinear term ($\beta w w_x$) in equation (1). This idea of balancing dispersion and nonlinearity is typical of any nonlinear time evolution equation that admits soliton solutions.

Solitons occur in multiple fields, including shallow- and deep-water waves, optical communications, Bose-Einstein condensates, and biological models. They possess universal characteristics and can be classified into several types, including water waves, sound waves, charge-density waves, matter waves, and electromagnetic waves (Stegeman & Mordechai Segev, 1999). Solitons are observed in several materials, including plasmas, Josephson junctions, and Polyacetylene (proteins and DNA) molecules, as shown in Figure 2 (Ya, 2013; Pérez-Cota *et al.*, 2016).

Solitons have many different uses. They may carry an electrical charge, and when charged, they travel along polymer chains that tend to curve. The study by Li et al. (2006) suggested that this characteristic has potential for future use in other fields, such as artificial muscles. Additionally, Borgese *et al.* (2015) reported that solitons can efficiently transmit substantial volumes of information over long distances, with little to no signal error. These solitons, with their peculiar dynamics and inherent stability, have been the subject of intense investigation over the years.

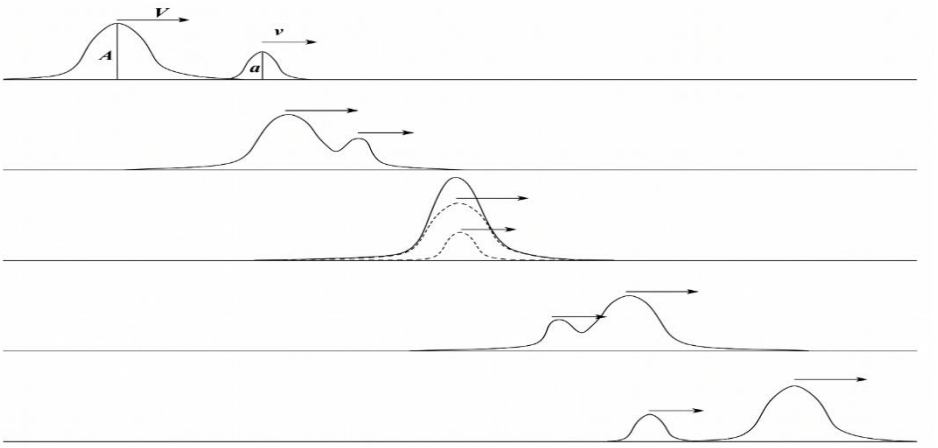
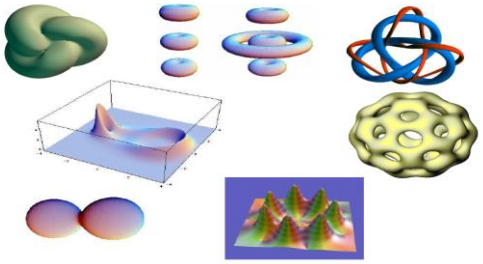
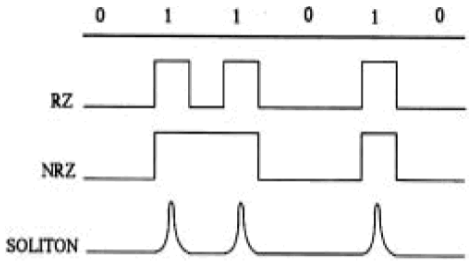


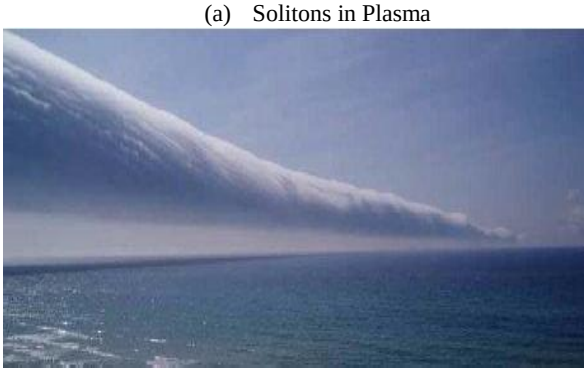
Figure 1: Collision of Two Solitons



(a) Solitons in Plasma

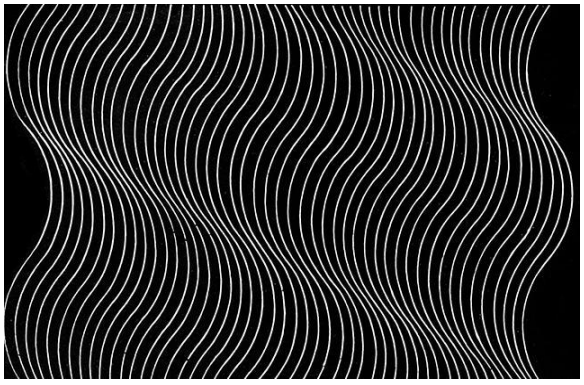


(b) Modulation Solitons in Fiber Optics



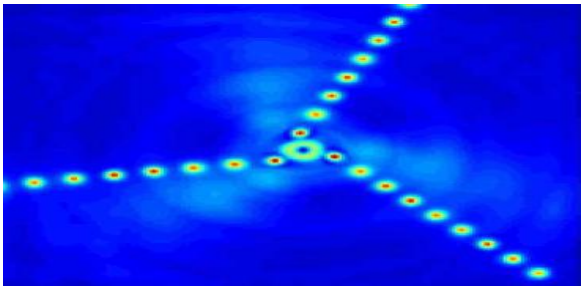
(c) Deep Water Soliton

(d) Shallow Water Solitons

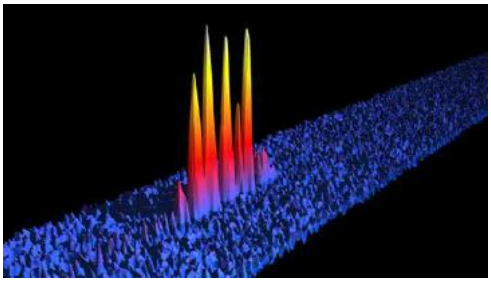


(e) Sound Wave Solitons

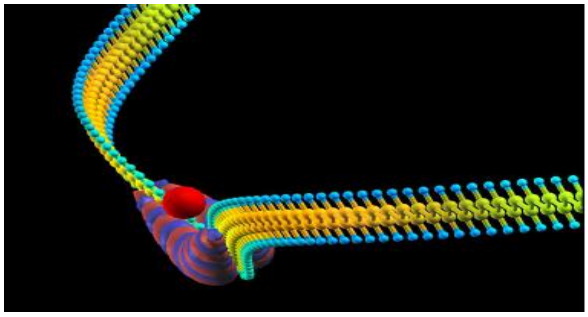
(f) Electromagnetic Soliton



(g) Electromagnetic Soliton (Laser Soliton in Fiber Optics)



(h) Solitons in Josephson Junction



(i) Soliton Wavefunction of Trans-polyacetylene Doped by a Counter Ion –kink Soliton

(j) Jupiter's Great Red Spot-Vortex Soliton

Figure 2: Multi-Solitons as they Occur in different Physical Areas (O'Neill, 2009; Menza *et al.*, 2011; Ya, 2013; María Soledad Jiménez *et al.*, 2016; Pérez-Cota *et al.*, 2016)

The KdV equation, initially formulated by Diederik Korteweg and Gustav de Vries in 1895 (Korteweg & de Vries, 1895), has since played a pivotal role in the study of nonlinear wave phenomena across various scientific disciplines. Its broad applicability is rooted in its capacity to describe and predict complex wave dynamics, ranging from shallow-water waves to electromagnetic pulses in fiber optics (Osman, 2017). This versatility has made it a cornerstone in the theoretical and computational exploration of wave phenomena. The discovery of soliton solutions to the KdV equation by Zabusky and Kruskal (1965) marked a significant milestone in the field. They observed that these solutions exhibit particle-like interactions, coining the term "Soliton". These properties of Solitons have piqued the curiosity of physicists, mathematicians, and engineers alike, as they hold the promise of a robust breakthrough in science and engineering.

The numerical exploration of soliton solutions to the KdV equation has been a topic of significant interest in computational physics. Numerical studies of solitons can provide invaluable insights into the behavior of these solitary wave packets under various conditions, illuminating their collisions and interactions (Alotaibi & Ismail, 2020). Moreover, it enables us to assess the accuracy and efficiency of various numerical methods and scientific computing languages for solving the KdV equation, a crucial consideration in the era of computational sciences.

The Leapfrog algorithm, a well-established numerical method, has been widely used to explore soliton solutions of the KdV equation. This algorithm's popularity stems from its numerical stability and computational efficiency, making it an ideal choice for simulating solitons (Shahrill *et al.,* 2015). Quite a lot of numerical methods have been proposed for the solution of partial differential equations, particularly nonlinear time evolution equations with soliton solutions, dating back as far as when Zabusky and Kruskal (1965) explained the FPU problem in terms of solitary wave solutions to the KdV equation using the Leapfrog scheme

$$w_j^{n+1} = w_j^{n-1} - \frac{\Delta t}{3\Delta x}\left(w_{j+1}^n + w_j^n + w_{j-1}^n\right)\left(w_{j+1}^n - w_{j-1}^n\right) - \delta^2 \frac{\Delta t}{\Delta x^3}\left(w_{j-2}^n - 2w_{j+1}^n + 2w_{j-1}^n - w_{j-1}^n\right) \quad (2)$$

which has a truncation error of $\{O(\Delta t)^2 + O(\Delta x)^2\}$ and requires that $\frac{\Delta t}{\Delta x}\left|-2w_{max} + \frac{1}{(\Delta x)^2}\right| \leq \frac{2}{3\sqrt{3}}$ to be stable. It is a standard upon which all other methods are compared. Many other researchers have continued the development of the finite difference schemes for the famous KdV equation, among which are the Greig and Morris (1976) "Hopscotch" scheme

$$w_j^{n+1} = w_j^n - \frac{\Delta t}{2\Delta x}\left(u_{j+1}^n - u_{j-1}^n\right) - \frac{\Delta t}{2(\Delta x)^3}\left(w_{j-2}^n - 2w_{j+1}^n + 2w_{j-1}^n - w_{j-1}^n\right),\ u = \frac{w}{2}, \quad (3)$$

with the stability criterion $\frac{\Delta t}{(\Delta x)^3} \leq \left|\frac{1}{(\Delta x)^2 2w_{max} - 2}\right|$.

The Goda (1977) implicit scheme

$$w_j^{n+1} = w_j^n - \frac{\Delta t}{\Delta x}\left(w_{j+1}^{n+1}\left(w_j^n + w_{j+1}^n\right) - w_{j-1}^{n+1}\left(w_j^n + w_{j-1}^n\right)\right) - \frac{\Delta t}{2(\Delta x)^3}\left(w_{j+2}^{n+1} - 2w_{j+1}^{n+1} + 2w_{j-1}^{n+1} - w_{j-2}^{n+1}\right) \quad (4)$$

which has a truncation error of $\{O(\Delta t) + O(\Delta x)^2\}$ and is unconditionally stable, meaning that stability is attained for any choice Δt.

Al-Khaled (2001) also developed a numerical scheme based on the Sinc-Galerkin method to approximate solutions to the KdV equation. Rageh *et al.* (2014) applied the Restrictive Taylor's technique to solve the KdV and Gardner equations numerically. Recently, Alotaibi and Ismail (2020) developed a numerical method based on Padé approximations for the KdV equation. Other numerical methods for solving the KdV equations are well documented in Johasen (2008) and Shahrill *et al.* (2015).

In this study, we modify the Leapfrog Scheme used by Zabusky and Krustal (1965) to compute numerically Multi-Solitons in the KdV equation and implement it in MATLAB and Python. The algorithm advances the solution in time using a staggered grid, thereby preserving important conservation properties of the KdV equation. The primary focus of this study is to explore the interaction of the multi-soliton solution of the KdV equation and to compare MATLAB and Python, which are commonly used scientific

computing languages. The analysis will focus on their application to the implementation of the Leapfrog scheme in KdV soliton simulations.

Numerical simulations of solitons are essential for verifying their theoretical behaviour. A key prediction is that solitons can pass through one another without changing shape, and numerical studies have consistently supported this prediction. However, small fluctuations in solitons after collisions have been observed in many simulations, and their nature remains an active research area (Hirota, 2004; Orapine et al., 2020; Kumar et al., 2022). In this context, selecting an appropriate programming language for implementing numerical algorithms is critical. MATLAB and Python are two popular options for scientific computing, each with its own strengths and weaknesses. Comparing these languages in the context of the Leapfrog algorithm for simulating KdV solitons can yield valuable insights into multi-soliton interactions and collisions, as well as the performance, accuracy, and efficiency of the algorithms. This comparison ultimately helps researchers make informed choices for their computational work.

This paper contributes to the ongoing discourse in computational physics by not only investigating the ripples generated by collisions or interactions among multi-solitons in the KdV equation but also providing a detailed assessment of the computational considerations associated with MATLAB and Python in numerical techniques for solving differential equations. By doing so, it extends our comprehension of the KdV equation, its soliton solutions, and the practical aspects of implementing numerical methods in computational physics.

METHODOLOGY

We look at the Taylor series approximation of the derivatives for the approximation of the initial value problem of the KdV equation (1).

To obtain the centered difference approximation, one first replaces the continuous variables x and t with discrete sets $\{x_i\}_0^N$ and $\{t_n\}_0^M$, respectively. These sets are so defined that x_i and t_n range over the intervals of interest. Usually, these are $-\infty < x < \infty$ and $t \geq 0$, but for numerical implementation, we must choose a finite set of intervals. We also suppose that the difference between successive values of x and t is the same. i.e. $\Delta x_i = x_i - x_{i-1} = h$ and $\Delta t_n = t_n - t_{n-1} = \tau$. Without loss of generality, we let $w(x_i, t_n) = w_i^n$.

Consequently, we obtained the centered difference approximations for first-order time and spatial derivatives and third-order spatial derivatives as:

$$w_t \approx \frac{w_i^{n+1} - w_i^{n-1}}{2\tau} + O(h^2) \quad (5)$$

$$w_x \approx \frac{1}{12h}(-w_{i+2}^n + 8w_{i+1}^n - 8w_{i-1}^n + w_{i-2}^n) + O(h^4) \quad (6)$$

$$w_{xxx} \approx \frac{w_{i-3}^n - 8w_{i-2}^n + 13w_{i-1}^n - 13w_{i+1}^n + 8w_{i+2}^n - w_{i+3}^n}{8h^3} + O(h^4) \quad (7)$$

Details of Taylor series approximations for derivatives are provided in Cho (2008) and Taylor (2016).

Numerical Computation Algorithm

We study the nonlinear KdV equation using the Leapfrog algorithm to discretize it on a grid, thereby enabling us to explore its behavior through numerical experiments.

Step 1*:* The first step is to describe the grid points. Because we need to observe waves propagating over long distances, as in physical applications, we make the computational domain a closed loop. This is done by choosing an interval from $x = 0$ *to* $x = L$, *and then making the system periodic by declaring that* $x = 0$ *and* $x = L$ are the same point, as would be the case in a circular channel. We subdivide this domain into N subintervals and let $h = \frac{L}{N}$ and $x_i = (i-1)h$, with the intention that the grid is cell-edge. Usually, such a cell-edge grid has $(N+1)$ points, but this is not because the last point $(i = (N+1))$ is just a repetition of the initial point: $x_{N+1} = x_1$ since our system is going to be periodic.

Step 2: We write the nonlinear KdV equation in a centered-difference approximation, namely the Leapfrog algorithm, so that the equation is expressed as a recurrence relation.

Step 3: Determine the stability of the leapfrog algorithm.

Step 4: Implement steps 1, 2, and 3 in MATLAB and Python to simulate multi-soliton interactions for rigorous investigation.

Leapfrogging through the KdV Equation

Here, we find the finite difference approximation of the nonlinear term of equation (1) modified from the usual approximations since the nonlinear term in (1) can also be written as $\beta w w_x = \frac{1}{2}\beta(w^2)_x$. If we let $u = w^2$, then we choose the fourth-order centered difference approximation (6) (Cho, 2008) (Taylor, 2016), for the order of accuracy considerations in the nonlinear term.

$$u_x = \frac{1}{12h}(-u_{i+2}^n + 8u_{i+1}^n - 8u_{i-1}^n + u_{i-2}^n) \quad (8)$$

Thus, the nonlinear term in (1) becomes

$$\beta w w_x = \frac{\beta}{24h}(-(w_{i+2}^n)^2 + 8(w_{i+1}^n)^2 - 8(w_{i-1}^n)^2 + (w_{i-2}^n)^2). \quad (9)$$

The fourth-order centered difference approximation (7) is also used for the third-order spatial derivative terms to achieve a balanced order of accuracy. Hence, the KdV equation (1) becomes a nonlinear system of algebraic equations

$$\frac{w_i^{n+1} - w_i^{n-1}}{2\tau} + \frac{\beta}{24h}(-(w_{i+2}^n)^2 + 8(w_{i+1}^n)^2 - 8(w_{i-1}^n)^2 + (w_{i-2}^n)^2) + \alpha\frac{w_{i-3}^n - 8w_{i-2}^n + 13w_{i-1}^n - 13w_{i+1}^n + 8w_{i+2}^n - w_{i+3}^n}{8h^3} = 0, \quad (10)$$

which can be written as

$$w_i^{n+1} = w_i^{n-1} - \frac{\beta\tau}{12h}(-(w_{i+2}^n)^2 + 8(w_{i+1}^n)^2 - 8(w_{i-1}^n)^2 + (w_{i-2}^n)^2) - \frac{\alpha\tau}{4h^3}(-w_{i+3}^n + 8w_{i+2}^n - 13w_{i+1}^n + 13w_{i-1}^n - 8w_{i-2}^n + w_{i-3}^n), \quad (11)$$

with a truncation error of $\{O(\tau)^2 + O(h)^4\}$. Equation (11) is our modified Zabusky-Kruskal Leapfrog algorithm for the KdV equation.

Linear Stability of the KdV Equation and Leapfrog Algorithm

We consider the linear KdV equation

$$w_t = -\alpha w_{xxx} \quad (12)$$

which is discretized using the Leapfrog algorithm as

$$w_i^{n+1} = w_i^{n-1} - r(-w_{i+3}^n + 8w_{i+2}^n - 13w_{i+1}^n + 13w_{i-1}^n - 8w_{i-2}^n + w_{i-3}^n), \; r = \frac{\tau\alpha}{4h^3}. \quad (13)$$

We denote the error in the approximate scheme as *and of course*; it must satisfy scheme (13). We use the Von Neumann stability analysis. The idea is to expand the error $\widehat{w}_i$ with a discrete Fourier series.

$$\widehat{w}_i = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \xi^{(k)} e^{jihk} \quad (14)$$

Where $j = \sqrt{-1}$ and N is the number of grid points normally in the periodic domain, say $[0, 2\pi)$.

Substituting (14) in the leapfrog scheme (13), we have that

$$\sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \xi^{(k+1)} e^{jihk} = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \xi^{(k)} e^{jihk}\left\{\frac{1}{\xi} - r(e^{-j3hk} - e^{j3hk} + 8[e^{j2hk} - e^{-j2hk}] - 13[e^{jhk} - e^{-jhk}])\right\} \quad (15)$$

Thus

$$\xi = \left\{\frac{1}{\xi} - r\left(e^{-j3hk} - e^{j3hk} + 8\left[e^{j2hk} - e^{-j2hk}\right] - 13\left[e^{-jhk} - e^{jhk}\right]\right)\right\}$$

$$\xi = \frac{1}{\xi} - j2r(-\sin(3hk) + 8\sin(2hk) - 13\sin(hk)) \quad (16)$$

The Von Neumann stability criterion states that the scheme (13) is stable if and only if the following modulus holds

$$|\xi| \leq 1 \quad (17)$$

Thus,

$$|j2r(-\sin(3hk) + 8\sin(2hk) - 13\sin(hk))| \leq 1 \quad (18)$$

It follows that,

$$r \leq \frac{1}{2(-\sin(3hk) + 8\sin(2hk) - 13\sin(hk))}$$

From (13), we have

$$\frac{\tau\alpha}{2h^3} \leq \frac{1}{2(-\sin(3hk)+8\sin(2hk)-13\sin(hk))} \quad (19)$$

And thus it follows that

$$\tau \leq \frac{h^3}{\alpha(-\sin(3hk)+8\sin(2hk)-13\sin(hk))} \quad (20)$$

The maximum value of

$$-\sin(3hk) + 8\sin(2hk) - 13\sin(hk) = 22 \quad (21)$$

Hence

$$\tau \leq \frac{h^3}{22\alpha} = \frac{h^3}{22}, \qquad \alpha = 1, \quad (22)$$

This is the stability condition we use to estimate the value of τ in equation (11); by implication, equation (11) is conditionally stable.

MATLAB Implementation: Setting the Boundaries in the Algorithm to Be Periodic

We have found a leapfrog algorithm (11) for the value of w at a given point in time and space, which contains the values of w for the previous $t-$values and the five closest $x-$values. The equation (11) can only be used to compute w_i^n for $3 < i \leq N-3$, but this is no problem since our system is periodic; we can set the system of algebraic equations to include the boundaries as follows:

$$\left.\begin{aligned}
w_1^{n+1} &= w_1^{n-1} - \tfrac{\beta\tau}{12h}(-(w_3^n)^2 + 8(w_2^n)^2 - 8(w_N^n)^2 + (w_{N-1}^n)^2) - \tfrac{\alpha\tau}{4h^3}(-w_4^n + 8w_3^n - 13w_2^n + 13w_N^n - 8w_{N-1}^n + w_{N-2}^n), \ i = 1\\
w_2^{n+1} &= w_2^{n-1} - \tfrac{\beta\tau}{12h}(-(w_4^n)^2 + 8(w_3^n)^2 - 8(w_1^n)^2 + (w_N^n)^2) - \tfrac{\alpha\tau}{4h^3}(-w_5^n + 8w_4^n - 13w_3^n + 13w_1^n - 8w_N^n + w_{N-1}^n), i = 2\\
w_3^{n+1} &= w_3^{n-1} - \tfrac{\beta\tau}{12h}(-(w_5^n)^2 + 8(w_4^n)^2 - 8(w_2^n)^2 + (w_1^n)^2) - \tfrac{\alpha\tau}{4h^3}(-w_6^n + 8w_5^n - 13w_4^n + 13w_2^n - 8w_1^n + w_N^n), \ i = 3\\
w_{N-2}^{n+1} &= w_{N-2}^{n-1} - \tfrac{\beta\tau}{12h}(-(w_N^n)^2 + 8(w_{N-1}^n)^2 - 8(w_{N-3}^n)^2 + (w_{N-4}^n)^2) - \tfrac{\alpha\tau}{4h^3}(-w_1^n + 8w_N^n - 13w_{N-1}^n + 13w_{N-3}^n - 8w_{N-4}^n + w_{N-5}^n), i = N-2\\
w_{N-1}^{n+1} &= w_{N-1}^{n-1} - \tfrac{\beta\tau}{12h}(-(w_1^n)^2 + 8(w_N^n)^2 - 8(w_{N-2}^n)^2 + (w_{N-3}^n)^2) - \tfrac{\alpha\tau}{4h^3}(-w_2^n + 8w_1^n - 13w_N^n + 13w_{N-2}^n - 8w_{N-3}^n + w_{N-4}^n), i = N-1\\
w_N^{n+1} &= w_N^{n-1} - \tfrac{\beta\tau}{12h}(-(w_2^n)^2 + 8(w_1^n)^2 - 8(w_{N-1}^n)^2 + (w_{N-2}^n)^2) - \tfrac{\alpha\tau}{4h^3}(-w_3^n + 8w_2^n - 13w_1^n + 13w_{N-1}^n - 8w_{N-2}^n + w_{N-3}^n), \ i = N\\
w_i^{n+1} &= w_i^{n-1} - \tfrac{\beta\tau}{12h}(-(w_{i+2}^n)^2 + 8(w_{i+1}^n)^2 - 8(w_{i-1}^n)^2 + (w_{i-2}^n)^2) - \tfrac{\alpha\tau}{4h^3}(-w_{i+3}^n + 8w_{i+2}^n - 13w_{i+1}^n + 13w_{i-1}^n - 8w_{i-2}^n + w_{i-3}^n), 3 < i < N-3
\end{aligned}\right\} \quad (23)$$

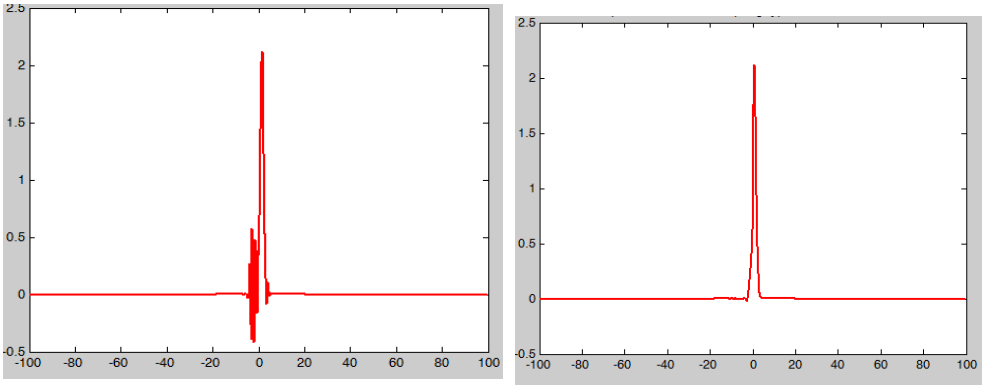
Again, the equation (11) can only be used to compute w_i^{n+1} for $1 \leq n \leq M-1$, whereas M is the final time, so to evaluate the algorithm at $n = 0$, we use a compromise of an FTCS to start the scheme implementation since the KdV equation is first order in time and so has only one initial condition.

$$w_i^1 = w_i^0 - \frac{\beta\tau}{24h}\left(-(w_{i+2}^0)^2 + 8(w_{i+1}^0)^2 - 8\left(w_{i-1}^0\right)^2 + (w_{i-2}^0)^2\right) - \frac{\alpha\tau}{2h^3}\left(-w_{i+3}^0 + 8w_{i+2}^0 - 13w_{i+1}^0 + 13w_{i-1}^0 - 8w_{i-2}^0 + w_{i-3}^0\right)$$

$i \leq N-3$ (24)

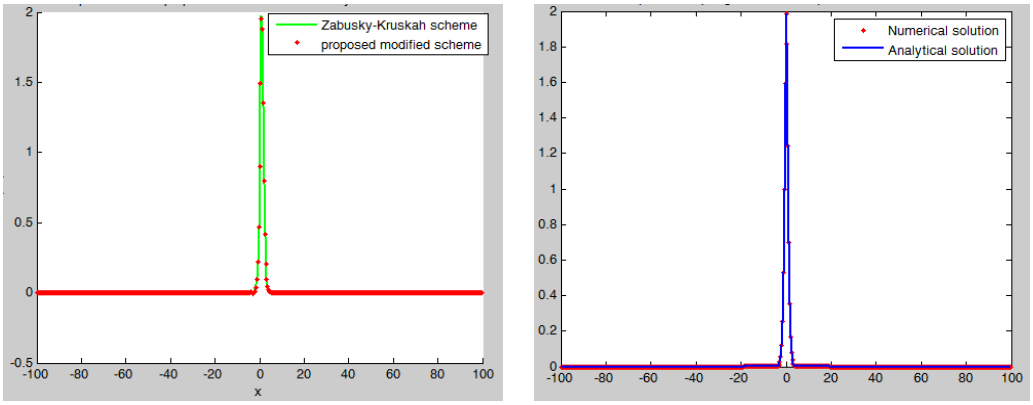
The same procedure of equation (23) is applied to equation (24) for boundary values. Thus, the algorithm implementation is complete to be coded in MATLAB since we have already estimated the value of $\tau = \Delta t$, how small or large enough that this algorithm will remain stable to avoid a blowup solution using linear stability condition in "section" 2.3 and illustrated in Figure 3, where $\tau = 0.01$ is shown to have given a better result.

We also compare the result of the new modified leapfrog algorithm to that of the Zubusky-Kruskal scheme and the exact solution at different time steps to verify the viability and the validity of the modified leapfrog scheme before going ahead to perform the multi-solitons interactions computation as captured in Figure 4. Figures 5-7 show the computed multi-solitons using the MATLAB implementation. The MATLAB code is given in Appendix A.



(a) at $\tau = 0.1$ (b) at $\tau = 0.01$ (better result)

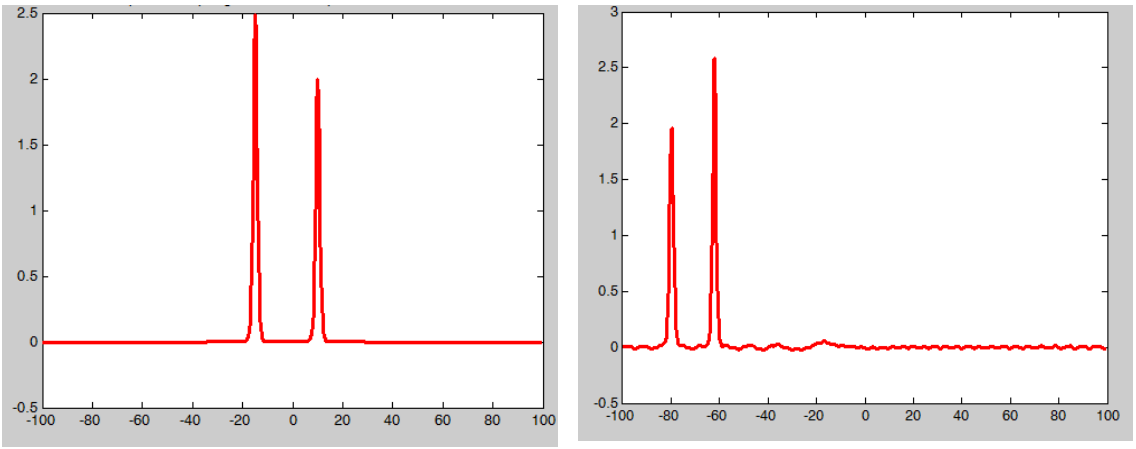
Figure 3: Linear Stability Condition



(a) New Modified Leapfrog Algorithm and the Zabusky-Kruskal Scheme

(b) New modified Leapfrog Algorithm with the Exact Solution

Figure 4: Comparison of the Newly Modified Leapfrog Algorithm with the Existing Scheme and Exact Solution



(a) Initial Propagation of Two Solitons

(b) Two after Collision or Interaction.

Figure 5: MATLAB Implementation of the Zabusky-Kruskal Scheme

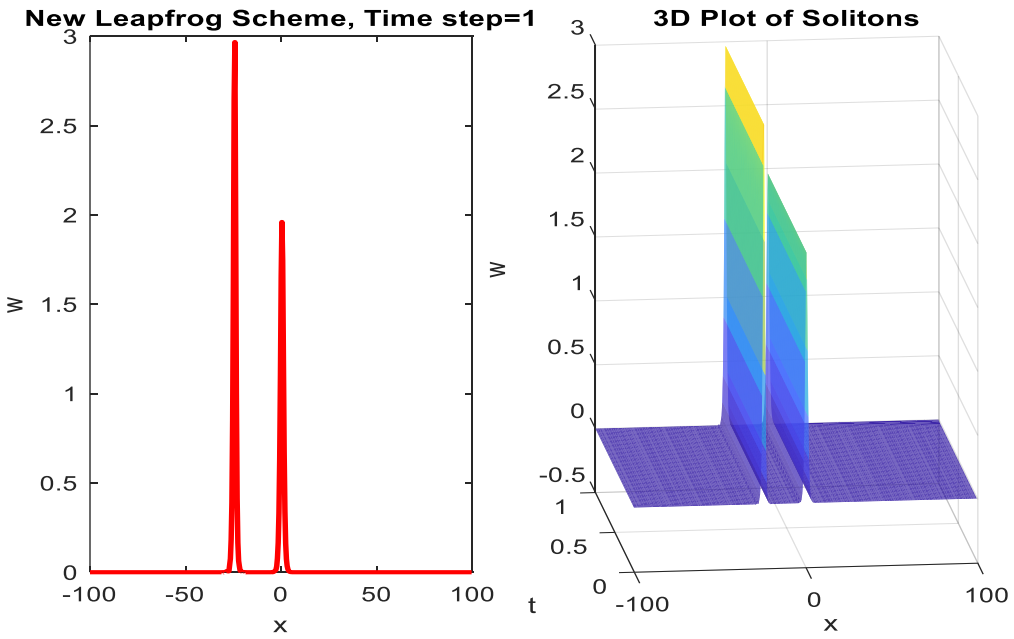


Figure 6: MATLAB Implementation: The Initial Stage of two Solitons Propagation, both in 2D and 3D, using the New Modified Leapfrog Algorithm

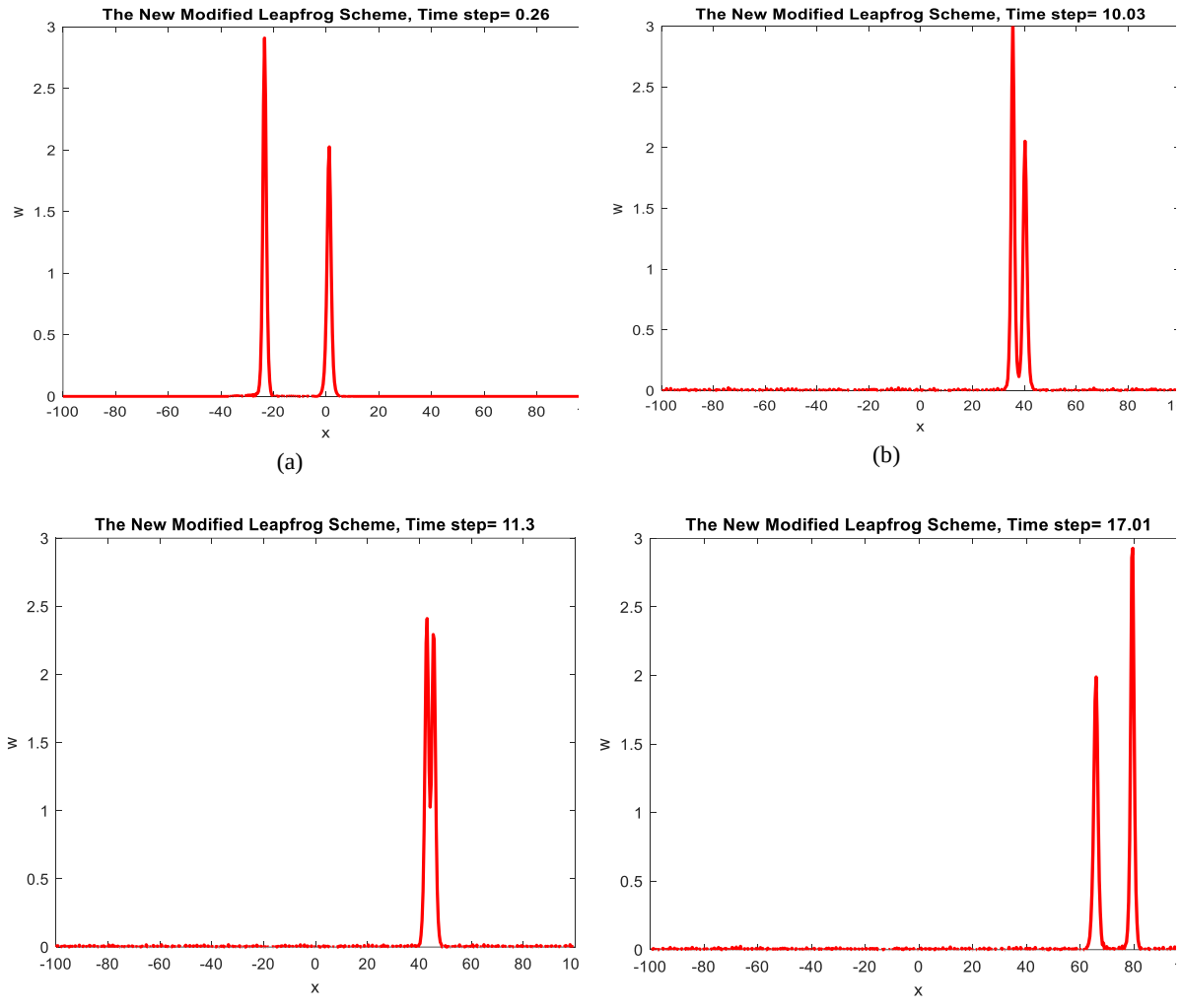


Figure 7: MATLAB Implementation: The Interaction of two Solitons Computed using the new Modified Leapfrog Algorithm, with Initial Profile $w = \frac{1}{2}\,k^2\,\mathrm{sech}^2\frac{k}{2}(x - x_0)\,, at\ \ \alpha = 1,\ \beta = 6,\ \tau = 0.01$

Python Implementation: Setting the Periodic Boundary Conditions

Setting periodic boundary conditions in Python is similar to that in MATLAB; however, the indexing differs between the two languages. Python indexing begins at 0, whereas MATLAB indexing begins at 1; therefore, we will review the periodic boundary conditions in equation (23) for the MATLAB implementation to align with the Python implementation. To do this, we recall that our cell-edge grid has $(N+1)$ points, but now $(i=N)$, since Python indexing starts at zero (0), and so the last point $(i = N)$ is a repetition of the initial point: $x_0 = x_N$ for the boundaries to be periodic. Thus, the equation (23) is modified for Python implementation as follows:

$$\left.\begin{aligned}
w_1^{n+1} &= w_1^{n-1} - \frac{\beta\tau}{12h}(-(w_3^n)^2 + 8(w_2^n)^2 - 8(w_0^n)^2 + (w_{N-1}^n)^2) - \frac{\alpha\tau}{4h^3}(-w_4^n + 8w_3^n - 13w_2^n + 13w_0^n - 8w_{N-1}^n + w_{N-2}^n),\ \ i = 1\\
w_2^{n+1} &= w_2^{n-1} - \frac{\beta\tau}{12h}(-(w_4^n)^2 + 8(w_3^n)^2 - 8(w_1^n)^2 + (w_0^n)^2) - \frac{\alpha\tau}{4h^3}(-w_5^n + 8w_4^n - 13w_3^n + 13w_1^n - 8w_0^n + w_{N-1}^n), i = 2\\
w_3^{n+1} &= w_3^{n-1} - \frac{\beta\tau}{12h}(-(w_5^n)^2 + 8(w_4^n)^2 - 8(w_2^n)^2 + (w_1^n)^2) - \frac{\alpha\tau}{4h^3}(-w_6^n + 8w_5^n - 13w_4^n + 13w_2^n - 8w_1^n + w_0^n),\ \ i = 3\\
w_{N-1}^{n+1} &= w_{N-1}^{n-1} - \frac{\beta\tau}{12h}(-(w_1^n)^2 + 8(w_0^n)^2 - 8(w_{N-1}^n)^2 + (w_{N-3}^n)^2) - \frac{\alpha\tau}{4h^3}(-w_2^n + 8w_1^n - 13w_0^n + 13w_{N-2}^n - 8w_{N-3}^n + w_{N-4}^n), i = N-1\\
w_0^{n+1} &= w_0^{n-1} - \frac{\beta\tau}{12h}(-(w_2^n)^2 + 8(w_1^n)^2 - 8(w_{N-1}^n)^2 + (w_{N-2}^n)^2) - \frac{\alpha\tau}{4h^3}(-w_3^n + 8w_2^n - 13w_1^n + 13w_{N-1}^n - 8w_{N-2}^n + w_{N-3}^n),\ \ i = N\\
w_i^{n+1} &= w_i^{n-1} - \frac{\beta\tau}{12h}(-(w_{i+2}^n)^2 + 8(w_{i+1}^n)^2 - 8(w_{i-1}^n)^2 + (w_{i-2}^n)^2) - \frac{\alpha\tau}{4h^3}(-w_{i+3}^n + 8w_{i+2}^n - 13w_{i+1}^n + 13w_{i-1}^n - 8w_{i-2}^n + w_{i-3}^n)\,, 3 < i < N-3
\end{aligned}\right\} \quad (25)$$

This is implemented together with equation (24) using the same periodic boundary conditions as in (25) and is thus ready for implementation in Python. The following Figures 8-10 present the result of the Python implementation of two soliton interactions and the Python code in Appendix B.

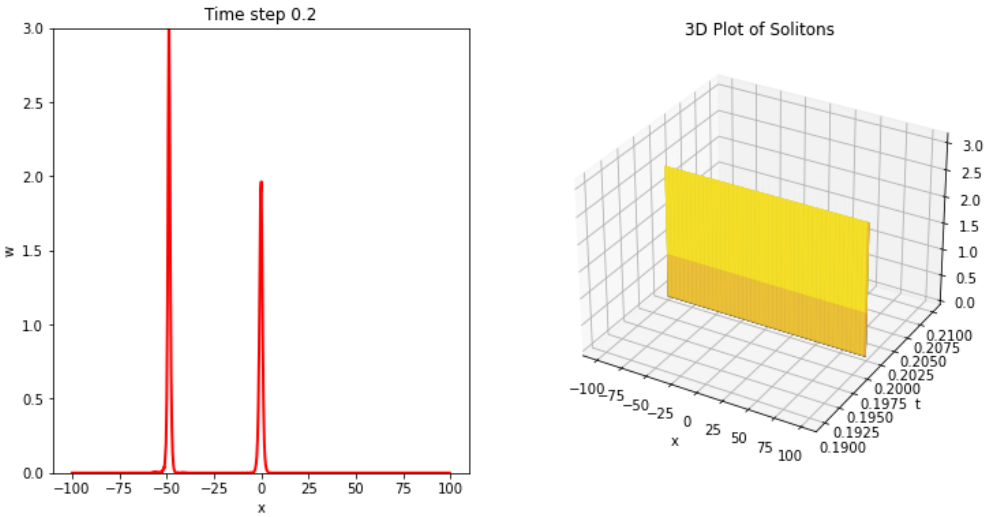
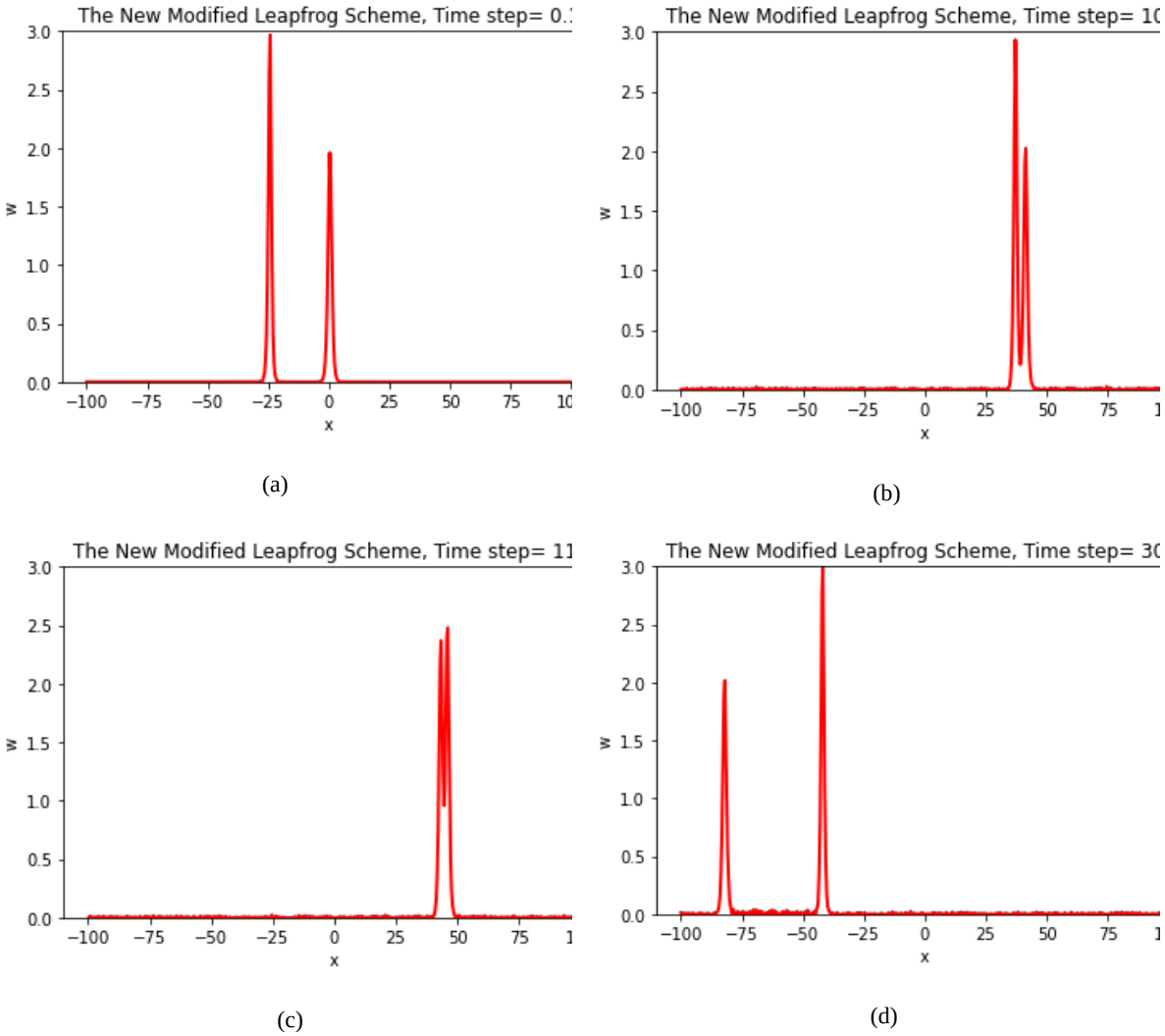


Figure 8: Python Implementation: The Initial Stage of Two Solitons Propagation, both in 2D and 3D, using the New Modified Leapfrog Algorithm



(a) (b)

(c) (d)

Figure 9: Python Implementation: The Interaction of Two Solitons Computed using the New Modified Leapfrog Algorithm, with Initial Profile $w = \frac{1}{2}\, k^2\, \mathrm{sech}^2 \frac{k}{2}(x - x_0)\,,\ at\ \alpha = 1,\ \beta = 6,\ \tau = 0.01$

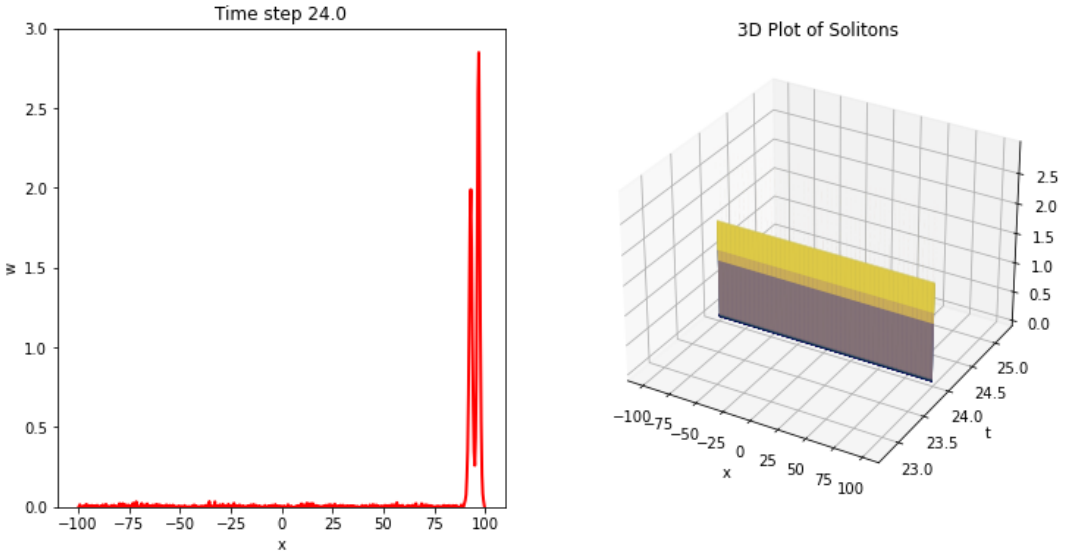


Figure 10: Python Implementation of 2D and 3D Plot: The Taller of the Two Solitons Leaves the Shorter One, just after an Interaction

Matlab and Python Performance Analysis

The selection of a programming language for scientific and computational work is inherently complex and shaped by specific needs, individual preferences, and available resources. While both MATLAB and Python offer features suitable for such tasks, this study focuses on the performance analysis of our unique leapfrog algorithm to evaluate which language is best suited to its implementation.

Program Execution Time

We measured how long it takes for the modified Leapfrog code to run in both MATLAB and Python using different numbers of grid points (N). The tests were repeated six times for each grid size, and the average time was calculated. MATLAB was generally faster, especially for smaller grids, while Python took more time as the grid size increased.

Program Execution Time

Time Step	MATLAB Execution Time at Each Test Run						Average Time	Python Execution Time at Each Test Run						Average Time
N	1	2	3	4	5	6	MATLAB	1	2	3	4	5	6	Python
1000	8.8844	12.5135	10.9732	11.5132	9.4026	9.1182	10.4009	12.5449	13.9207	12.5905	16.1717	12.7382	12.4667	13.4055
3000	15.845	15.4232	17.2036	14.352	15.524	12.7132	15.1770	42.4793	42.2719	45.6545	37.2395	38.523	38.8057	40.8290
5000	39.059	19.6675	24.9239	19.2305	22.837	25.6942	25.2353	64.6434	62.9634	65.3087	64.8588	63.7374	62.816	64.0546
7000	25.547	23.842	29.6118	31.2815	26.629	29.3196	27.7052	90.7166	89.2276	88.1936	92.5116	88.8629	87.6573	89.5283
9000	33.717	30.1675	30.7511	32.4836	30.398	30.1566	31.2789	119.721	115.8112	114.1669	145.9925	146.8506	147.6289	131.6951

Figure 11: MATLAB and Python Execution Time at different Times and Steps

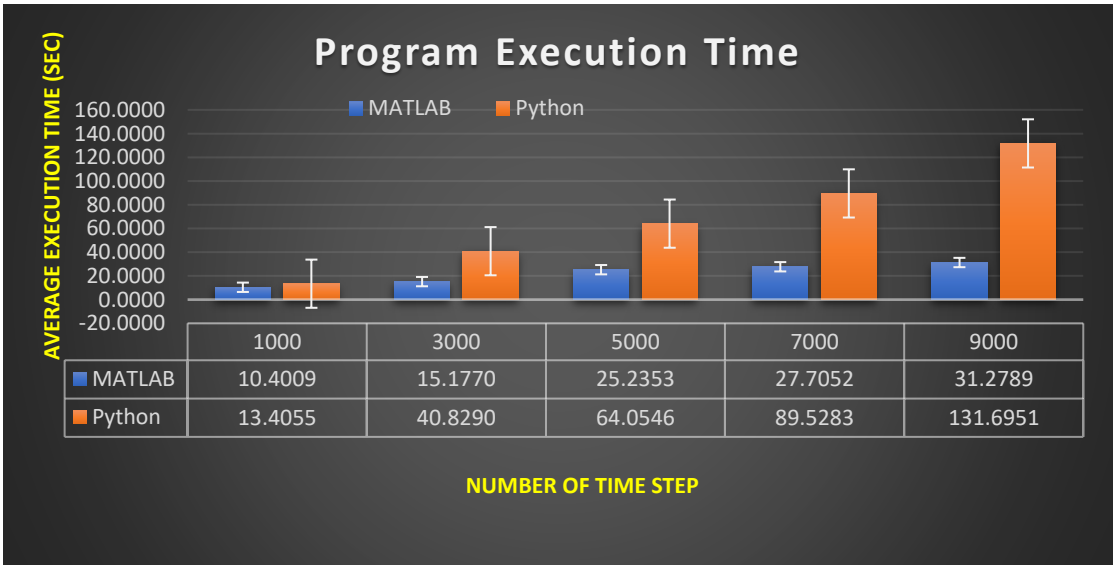


Figure 12: Bar Graph Comparing MATLAB and Python Execution time at Different Times and Steps

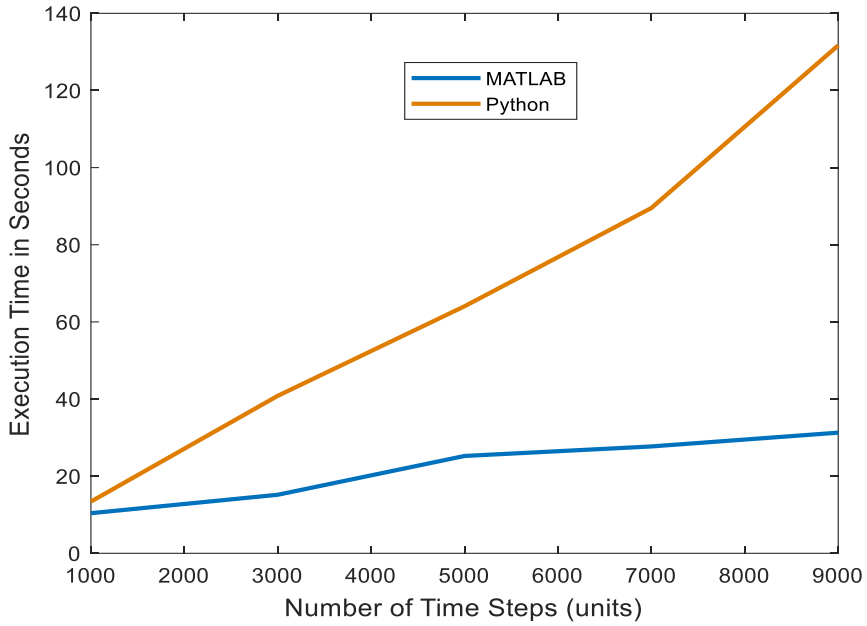


Figure 13: Line Graph Comparing MATLAB and Python Execution Time at different Time Steps

Discussion

In this study, we have studied the soliton solution of the KdV equation. Solitons are typically stable, propagating solutions of nonlinear evolution models, characterized by a balance between a dispersive term (which causes a traveling wave to spread) and a nonlinear term (which causes a traveling wave to steepen). We have performed several numerical computations. We modified the Zabuky-Kruskal finite-difference scheme, which yields improved stability, as illustrated in Figures (3- 4a), and reduced truncation error, as the exact solution and numerical results are compared at each time step in the exact simulation, as indicated in Figure (4b). The agreement between analytical and numerical solutions is excellent.

The linear stability of the KdV equation and the modified Leapfrog algorithm are investigated, and the stability condition of the new Leapfrog scheme for simulating multi-solitons is estimated. We note here, however, that $\tau = 0.01$ is estimated sufficiently small because of stability issues (to avoid blowup solutions which we have seen for larger time step τ as demonstrated in Figure (3)) because of the nonlinear term in the KdV equation. Solitons in the numerical simulations move either left to right or right to left, depending on the sign (positive or negative) in the nonlinear term in the KdV equation.

In Figures (6 & 8), the algorithm was implemented in MATLAB and Python for 2D and 3D two-soliton propagation, with initial profile $\frac{1}{2}k^2\,\mathrm{sech}^2\frac{k}{2}(x - x_0)$ computed at $\alpha = 1,\ \beta = 6\ and\ k_1 = 6,\ k_2 = 4,$ and initial points of propagation $x_{01} = 25,\ x_{02} = 0$. Figure 7 presents the MATLAB implementation of the leapfrog algorithm for multi-soliton interactions. At the same time, Figures (9-10) represent the Python implementation of the Leapfrog algorithm for multi-soliton computations and interactions. In the study of the interactions of two solitons, they propagated from left to right with the taller one moving faster. Solitons in our simulations propagate to different locations: the taller one is located farther left, but it eventually moves faster and overtakes the ones with smaller amplitudes. It is also observed that as two solitons collide with each other, they momentarily form a single soliton pulse as represented in Figures (7c & 9c). Afterward, they separate and continue with their original identities. It is also noticed that during interactions of solitons in all the simulations carried out, the amplitude of the supposed single soliton pulse formed by two or more solitons is observed to be smaller than the amplitude of the larger soliton between the interactions. For example, in Figure (7a), the larger soliton amplitude is about 3.0 units, while the amplitude of the single-soliton pulse formed is 2.5 units, but regains its height or amplitude after interactions. This is also evident in Figure (9a). Again, there is a phase shift after interactions, since the smaller soliton in front moves to the rear and the larger soliton moves farther ahead, unlike in linear waves. This confirms that solitons do not obey the superposition principle but interact nonlinearly with each other.

Performance Comparison between MATLAB and Python

We ran both implementations on the same hardware with identical grid parameters (N = 512 points, $\Delta x \approx 0.1$, Δt chosen to satisfy the stability condition from section 2.3).

i. ***Accuracy****:* Both codes produced nearly identical results, with L2-norm errors against the exact solution below 10^{-4} after long simulation times. The modified fourth-order scheme slightly outperformed the original Zabusky-Kruskal version in reducing post-collision oscillations.

ii. ***Efficiency****:* MATLAB was faster for smaller grids due to its optimized matrix operations and built-in vectorization. Python (using NumPy) was competitive and sometimes faster for larger grids when leveraging just-in-time compilation (e.g., via Numba). Python offers better readability and easier integration with data analysis libraries like Matplotlib and SciPy.

iii. ***Ease of Implementation****:* Python's zero-based indexing required minor adjustments to boundary handling (as shown in equation 25), but the code is more portable and open-source friendly. MATLAB excels in rapid prototyping with excellent visualization tools.

These findings align with general observations in scientific computing: MATLAB is often preferred for quick numerical experiments in academia, while Python provides greater flexibility for larger projects and collaboration.

Small discrepancies between numerical and exact solutions after collisions are expected in finite-difference methods but remain negligible and do not affect the key physical conclusions.

CONCLUSION

This study successfully implemented a modified Leapfrog finite-difference scheme for solving the Korteweg-de Vries equation in both MATLAB and Python. The numerical results clearly demonstrate the existence and stability of multi-soliton solutions. Solitons interact in a remarkable way—they collide and pass through each other while preserving their amplitude, speed, and shape, only experiencing a phase shift. This confirms long-standing theoretical expectations and highlights the power of simple numerical methods in revealing deep nonlinear phenomena. The comparative analysis shows that both languages are suitable for this type of work. Choice depends on user needs: MATLAB for speed in prototyping and plotting, Python for openness, scalability, and modern data science workflows. Future work could extend this to higher-order schemes, adaptive grids, or parallel implementations for more demanding simulations. The Leapfrog approach remains a reliable, conservation-preserving tool for studying nonlinear waves, bridging analytical theory with practical computation. Our results add to the body of evidence supporting the use of numerical experiments to explore soliton dynamics in physics and engineering applications.

REFERENCES

Al-Khaled Kamel. (2001). Sinc numerical solutions for solitons and solitary waves. *Journal of Computational and Applied Mathematics*, *130*(1-2), 283–292. https://doi.org/10.1016/s0377-0427(99)00376-3

Alotaibi, F., & Ismail, M. S. (2020). Numerical Solution of Kortweg-de Vries Equation. *Applied Mathematics*, *11*(04), 344–362. https://doi.org/10.4236/am.2020.114025

Borgese, G., Vena, S., Pantano, P., Pace, C., & Bilotta, E. (2015). Simulation, Modeling, and Analysis of Soliton Waves Interaction and Propagation in CNN Transmission Lines for Innovative Data Communication and Processing. *Discrete Dynamics in Nature and Society*, *2015*, 1–13. https://doi.org/10.1155/2015/139238

Cho, H. K. (2008). INTRODUCTION TO NUMERICAL ANALYSIS. In *SNO courseware* (pp. 1–48). Nuclear Thermal Hydraulic Engineering Lab: Department of Nuclear Engineering, Seoul National University. https://ocw.snu.ac.kr/sites/default/files/NOTE/Lecture%2008_0.pdf

Goda, K. (1977). Numerical studies on recurrence of the Korteweg-de Vries equation. *Journal of the Physical Society of Japan*, *42*, 1040–1046.

Greig, I. S., & Morris, J. (1976). A Hopscotch method for the Korteweg-de-Vries equation. *Journal of Computational Physics*, *20*(1), 64–80. https://doi.org/10.1016/0021-9991(76)90102-9

Haruna, A., Olatunbosun, R. O., Ibiyemi, A. A., & Awodugba, A. O. (2018). Analysis of electromagnetic radiation for four different transmission power stations areas of Nigeria. *FUDMA Journal of Sciences*, *2*(1), 48–53.

Helfrich, K. R., & Melville, W. K. (2006). LONG NONLINEAR INTERNAL WAVES. *Annual Review of Fluid Mechanics*, *38*(1), 395–425. https://doi.org/10.1146/annurev.fluid.38.050304.092129

Hirota, R. (2004). *The Direct Method in Soliton Theory*. Cambridge University Press. https://doi.org/10.1017/cbo9780511543043

Johansen, B. R. (2008). *Numerical Investigations of the Korteweg-de Vries (KdV) Equation*. MSc. Thesis). School of Postgraduate Studies, Memorial University of Newfoundland, St. John's Newfoundland-Canada. https://research.library.mun.ca/8921/

Korteweg, D. J., & de Vries, G. (1895). XLI. On the change of form of long waves advancing in a rectangular canal, and on a new type of long stationary waves. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *39*(240), 422–443. https://doi.org/10.1080/14786449508620739

Kumar, D., Nuruzzaman, Md., Paul, G. C., & Hoque, A. (2022). Novel localized waves and interaction solutions for a dimensionally reduced (2 + 1)-dimensional Boussinesq equation from N-soliton solutions. *Nonlinear Dynamics*, *107*(3), 2717–2743. https://doi.org/10.1007/s11071-021-07077-9

Li, J., Xi, L., Clemens, F., & Yip, S. (2006, July 6). *Solitons could power molecular electronics and artificial muscles*. Phys.org; Proceedings of the National Academy of Sciences (PNAS). http://www.physorg.com/news71409967.html

María Soledad Jiménez, Poveda, G., & Carrera, C. (2016). Estudio y Simulación de la propagación de Solitones en una Fibra Óptica Monomodo. *Revista Politécnica: DOAJ (DOAJ: Directory of Open Access Journals)*, *38*(1).

Menza, L., Debussche , A., & Barton-Smith, M. (2011). *Physical context and mathematical model Simulation of a stochastic model: Numerical methods for the stochastic Schödinger equation*. Laboratoire de Math´ematiques - Universit´e de Reims. https://esnt.cea.fr/Images/Page/27/dimenza.pdf

O'Neill, A. I. (2009, June 2). *Holland*. Astroengine.com. https://astroengine.com/tag/holland/

Orapine, H. O., Ayankop-Andi, E., & Ibeh, G. J. (2020). Analytical and Numerical Computations of Multi-Solitons in the Korteweg-de Vries (KdV) Equation. *Applied Mathematics*, *11*(07), 511–531. https://doi.org/10.4236/am.2020.117037

Osman, M. S. (2017). Analytical study of rational and double-soliton rational solutions governed by the KdV–Sawada–Kotera–Ramani equation with variable coefficients. *Nonlinear Dynamics*, *89*(3), 2283–2289. https://doi.org/10.1007/s11071-017-3586-y

Pérez-Cota, F., Smith, R. J., Moradi, E., Marques, L., Webb, K. F., & Clark, M. (2016). High-resolution 3D imaging of living cells with sub-optical wavelength phonons. *Scientific Reports*, *6*(1). https://doi.org/10.1038/srep39326

Rageh, T. M., Salem, E., & El-Salam F. A. (2014). Restrictive Taylor Approximation for Gardner and KdV Equations. *International Journal of Advances in Applied Mathematics and Mechanics*, *1*(13).

Shahrill, M., Chong, M. S. F., & Mohd Nor, H. N. H. (2015). Applying Explicit Schemes to the Korteweg-de Vries Equation. *Modern Applied Science*, *9*(4). https://doi.org/10.5539/mas.v9n4p200

Stegeman, G. I., & Mordechai Segev. (1999). Optical Spatial Solitons and Their Interactions: Universality and Diversity. *Science*, *286*(5444), 1518–1523. https://doi.org/10.1126/science.286.5444.1518

Taylor, C. R. (2016). *Finite difference coefficients calculator*. MIT. https://web.media.mit.edu/~crtaylor/calculator.html

Ya, S. (2013). *Introduction to Solitons* (pp. 1–40). Institute of Theoretical Physics and Astronomy: University of Oldenburg and BSU Minsk. http://www.tfai.vu.lt/files/shnir/Lecture1.pdf

Zabusky, N. J., & Kruskal, M. D. (1965). Interactions of "solitons" in a collisionless plasma and the recurrence of initial states. *Physics Review Letters*, *15*(240).