Solitons - The KdV Equations

Arshia Akhtarkavan

December 2025

Abstract

In this Final Term Project, we examine the soliton solutions of the Korteweg-de Vries (KdV) equation by reducing the nonlinear partial differential equation to an ordinary equation using a traveling wave ansatz. The resulting third-order ODE is turned into a system of first-order ODEs, and is solved numerically using a fourth-order Runge-Kutta (RK4) scheme. A conserved quantity (the constant of motion) is derived and used to test the accuracy of the numerical integration. Next, the traveling wave solution is reconstructed, giving us the full soliton profile $\phi(x,t)$, which perfectly matched our expected shape of a soliton.
To validate the numerical solutions, convergence tests are performed by examining the error in the constant of motion as the step size is decreased. The results show a fourth-order convergence, which is consistent with the RK4 method. A self-convergence test is also performed due to the lack of an exact analytical solution, which further confirmed the correctness of our numerical implementation. Richardson extrapolation is then used to estimate the numerical error, which showed that the solutions are so accurate that they are near machine precision at sufficiently small step sizes.
All of the code used in this project has been submitted alongside this report. Additionally, all of the files used for this project is publicly available on GitHub at `https://github.com/AaKavan/SolitonDynamics.git`.

Contents

1 Introduction **2**

2 Mathematical Formulation **2**

3 Numerical Methods **4**
- 3.1 Structure of the Code . . . 4
- 3.2 Utilizing Symmetry . . . 4
- 3.3 Step Size . . . 4
- 3.4 Storing the results . . . 5

4 Numerical Results **5**
- 4.1 Forward Integration of the ODE and the Construction of the Full Solution . . . 5
- 4.2 $C(X)$ and error in the Constant of Motion . . . 6

5 Traveling Wave Reconstruction **7**

6 Convergence of the Constant of Motion **8**

7 Self-Convergence of Solutions **9**

8 Richardson Extrapolation **10**

9 Conclusions **11**

1 Introduction

A soliton is a special solution of a nonlinear partial differential equation (PDE) wave equation that has the following three properties [2]:

1. They are of permanent form;
2. They are localized within a region;
3. They can interact with other solitons and emerge from the collision unchanged, except for a phase shift.

The Korteweg - de Vries (KdV) equation first discovered in 1895 by Diederik Korteweg and Gustav de Vries is a PDE that models the solitary waves that were first observed by John Scott Russel in the 19th century. The standard form of the KdV equation is

$$\phi_t - 6\phi\phi_x + \phi_{xxx} = 0 \tag{1}$$

where $\phi = \phi(x,t)$ is the height of the wave at position x and time t.

$$\phi_t = \frac{\partial \phi}{\partial t}, \qquad \phi_x = \frac{\partial \phi}{\partial x}, \qquad \phi_{xxx} = \frac{\partial^3 \phi}{\partial x^3} \tag{2}$$

2 Mathematical Formulation

To solve this PDE, we can first begin by looking for wave-like solutions of the form

$$\phi(x,t) = f(x - ct - a) = f(X), \qquad X \coloneqq x - ct - a \tag{3}$$

We can now plug this initial guess (also known as an ansatz) into the KdV equation (1). Since $X = x - ct - a$, we have:

$$\frac{\partial X}{\partial x} = 1, \quad \& \quad \frac{\partial X}{\partial t} = -c \tag{4}$$

Then by applying the chain rule, we can find each of the derivatives as follows:

$$\phi_x = \frac{\partial \phi}{\partial x} = \frac{\partial f}{\partial x} = \frac{\partial f}{\partial X}\frac{\partial X}{\partial x} = f'(X) \tag{5}$$

$$\phi_t = \frac{\partial \phi}{\partial t} = \frac{\partial f}{\partial t} = \frac{\partial f}{\partial X}\frac{\partial X}{\partial t} = -cf'(X), \tag{6}$$

$$\phi_{xxx} = \frac{\partial^3 \phi}{\partial x^3} = \frac{\partial^3 f}{\partial x^3} = \frac{\partial^3 f}{\partial X^3}\left(\frac{\partial X}{\partial x}\right)^3 = f'''(X) \tag{7}$$

Substituting these into (1):

$$\phi_t - 6\phi\phi_x + \phi_{xxx} = 0 \implies -cf'(X) - 6f(X)f'(X) + f'''(X) = 0 \tag{8}$$

$$\implies f'''(X) = \big(c + 6f(X)\big)f'(X) \tag{9}$$

Now, we can actually reduce this equation to a first-order system of equations. Observe that the transformation $X \mapsto -X$ leaves this equation invariant. This is due to the symmetry of the traveling wave equation that we obtained, since under the $X \mapsto -X$ transformation, we have:

$$\frac{d}{dX} \mapsto -\frac{d}{dX} \qquad \text{and} \qquad \frac{d^3}{dX^3} \mapsto -\frac{d^3}{dX^3} \tag{10}$$

So both sides of the equation change sign, and we end up with the original equation, just with a $-X$ instead of X everywhere. Integrating both sides of equation (9), we find that

$$\int f'''(X)dX = \int (c + 6f(X))f'(X)dX \implies f''(X) + C_1 = (c\int f'(X)dX + 6\int f(X)f'(X)dX) \tag{11}$$

$$\int f'(X)dX = f(X) \quad \& \quad \int f(X)f'(X)dX = \int f(X)dX = \frac{f(X)^2}{2} \tag{12}$$

$$\implies f''(X) + C_1 = cf(X) + 3f(X)^2 + C_2 \implies C_2 - C_1 = C = f''(X) - cf(X) - 3f(X)^2 \tag{13}$$

Thus we find that

$$C = f''(X) - cf(X) - 3f(X)^2 = f''(X) - (c + 3f(X))f(X) \tag{14}$$

is a constant of the motion. This will be useful in validating the accuracy of our numerical integration of (9) later on. For the simplicity of this project, we set $c = 1$. We have the initial conditions

$$f(0) = -\frac{1}{2}, \qquad f'(0) = \frac{df}{dX}\Big|_{X=0} = 0, \qquad f''(0) = \frac{d^2f}{dX^2}\Big|_{X=0} = \frac{1}{4}, \tag{15}$$

And so by plugging these values into our constant of motion, we get

$$C(0) = f''(0) - (1 + 3f(0))f(0) = \frac{1}{4} - (1 + 3(\frac{-1}{2}))(\frac{-1}{2}) = \frac{1}{4} - (-\frac{1}{2})(-\frac{1}{2}) = \frac{1}{4} - \frac{1}{4} = 0 \tag{16}$$

So $C(0) = C \equiv 0$ for this project.
To numerically solve this ODE using the 4th-order Runge-Kutta method (RK4), we must first reduce it to a system of first-order equations.
Define

$$f_1 \coloneqq f(X) = f, \qquad f_2 \coloneqq \frac{df}{dX}, \qquad f_3 \coloneqq \frac{d^2f}{dX^2} \tag{17}$$

Which gives us the system of equations

$$\begin{cases} f_1' = f_2 \\ f_2' = f_3 \\ f_3' = (1 + 6f_1)f_2 \end{cases} \tag{18}$$

with

$$\begin{cases} f_1(0) = -\frac{1}{2} \\ f_2(0) = 0 \\ f_3(0) = \frac{1}{4} \end{cases} \tag{19}$$

And we can integrate forward for $X > 0$ and from the symmetry conditions (the invariance of X under the transformation $X \mapsto -X$, find the backwards solutions as well. Now let us write this system of differential equations as a vector ODE. Thus

$$\frac{d\vec{u}(X)}{dt} = \vec{F}(X, \vec{u}(X)), \qquad \text{where} \qquad \vec{u} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}, \qquad \vec{F}(X, \vec{u}) = \begin{bmatrix} f_2 \\ f_3 \\ (1 + 6f_1)f_2 \end{bmatrix} \tag{20}$$

And the initial condition vector of

$$\vec{u}(0) = \begin{bmatrix} -\frac{1}{2} \\ 0 \\ \frac{1}{4} \end{bmatrix} \tag{21}$$

Now for a step size of $h > 0$ and a grid of $X_n = X_0 + nh$, we can apply the RK4 method:

$$\vec{k_1} = \vec{F}(X_n, \vec{u}_n) \tag{22}$$

$$\vec{k_2} = \vec{F}(X_n + \frac{h}{2}\ ,\ \vec{u}_n + \frac{h}{2}\vec{k_1}) \tag{23}$$

$$\vec{k_3} = \vec{F}(X_n + \frac{h}{2}\ ,\ \vec{u}_n + \frac{h}{2}\vec{k_2}) \tag{24}$$

$$\vec{k_4} = \vec{F}(X_n + h\ ,\ \vec{u}_n + h\vec{k_3}) \tag{25}$$

$$\vec{u}_{n+1} = \vec{u}_n + \frac{h}{6}(\vec{k_1} + 2\vec{k_2} + 2\vec{k_3} + \vec{k_4}) \tag{26}$$

These equations will now be solved numerically by implementing a 4th-order Runge-Kutta (RK4) integrator.

3 Numerical Methods

To solve this system of ODEs, instead of a single variable in the RK4 method, which is what was used during the homework assignments, we need to write the code in a "vectorized" way so that the integrator can work.

3.1 Structure of the Code

The C program consists of three main components:

- A right-hand-side (denoted by rhs) function

  ```c
  void rhs(double X, double u[], double F[])
  ```

  This function computes
  $$\vec{F}(X,\vec{u}) = (u_2, u_3, (1+6u_1)u_2) \tag{27}$$

- A general RK4 integrator

  ```c
  void rk4(double X, double u[], double h)
  ```

  Which serves to calculate $\vec{u}_{n+1}$, as written in equation 26 by calling the rhs() function.

- A typical driver, which serves to take the steps forward over the integration domain. This loop would integrate from
  $$X = 0 \qquad \text{to} \qquad X_{\max} = 20 \tag{28}$$
  and would save the values of $f(X)$, $f'(X)$, and $f''(X)$ at every point in the grid in an array[1].

3.2 Utilizing Symmetry

Since we showed that $f(X) = f(-X)$, to integrate over the entire interval of $[-20, 20]$, all we have to do is compute the forward solution for $X > 0$, and reflect the values across the y-axis (except for $X = 0$, since $f(0) = f(-0)$). This not only helps with computational costs but also makes it so that we only need a forward integrator function implemented.

3.3 Step Size

To determine an appropriate integration step size h, I first started with a step size of $h = 0.001$, which corresponds to
$$N = \frac{20}{0.001} = 20,000 \ \ \text{steps} \tag{29}$$
This value of h was chosen arbitrarily and, as we will discuss shortly, turned out to be very accurate. The main way of ensuring accuracy for our solutions is by using the constant of motion (described in 14) and by the fact that analytically, $C(X) = 0$ everywhere. For each step size, we can calculate
$$\delta C = |C(X) - C| = |C(X) - 0| = |C(X)| \tag{30}$$
for decreasing step sizes h. Note that the reason that an absolute error was used instead of the more typically used relative error was due to the fact that with the given initial conditions, $C = 0$ and thus means that we can't perform a division by it. However, an absolute error should be just as good for determining the accuracy.
In the following section, I will describe the results of these calculations, with plots made for demonstration purposes. As we will see, the solutions were solved with very high levels of accuracy.

[1]To keep the code simple and more aligned with the provided templates, I did not use vectors, which are dynamic arrays that are commonly used in C++. I instead used simple, fixed-size arrays for the entirety of this project, including this part.

3.4 Storing the results

After each iteration of the code, all of the values, including X, $f(X)$, $f''(X)$, $C(X)$, and the absolute error $\delta C(X)$, get written to an output file. Finally, these written results will be used to create plots, which are shown in the next part of this document. All of the code for plotting was implemented in Python with the help of the *Numpy* and *Matplotlib* libraries.
At this point, the essential numerical methods used in this project have been described. Although the later sections have code that could be explained in more detail, it is not done here in order to keep the report focused on the numerical results and their interpretations. All source code submitted alongside this report (and publicly available at [1]) is documented well with explanatory comments throughout it.

4 Numerical Results

4.1 Forward Integration of the ODE and the Construction of the Full Solution

The first of these figures, Figure 1, is the numerical solution for $f(X)$ for $X > 0$, which was created as outlined in Part 3. As we'd expect, it starts from the initial value of -0.5 and smoothly rises to zero as X increases.

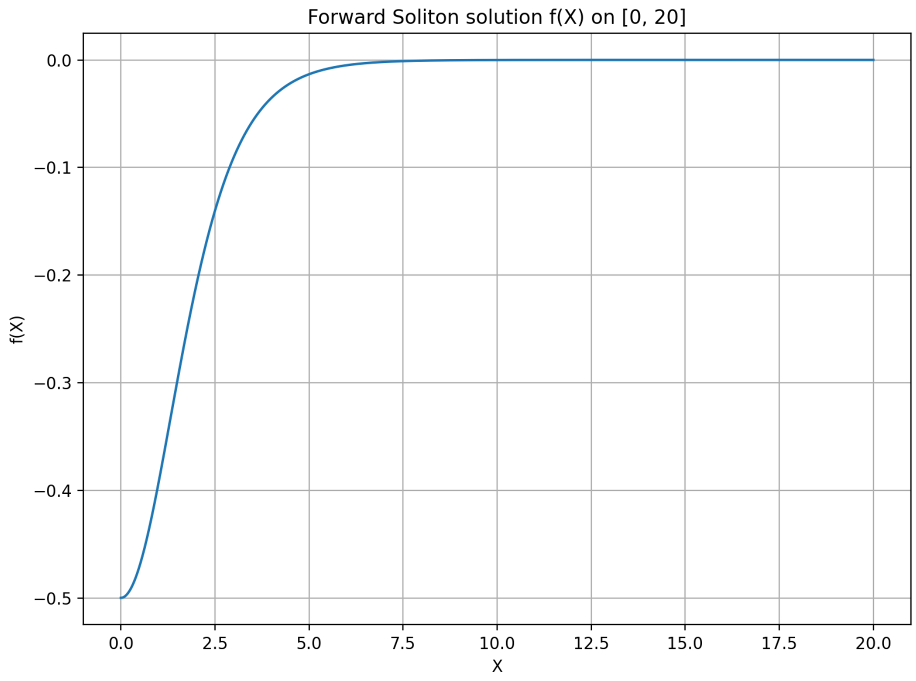

Figure 1: Forward Soliton solution $f(X)$ on $[0, 20]$

Next, by using the symmetry property of $f(X)$ and reflecting across the y-axis, we get Figure 2, which is the full numerical solution on $[-20.20]$. This is exactly the structure that a stationary soliton with our given assumptions should have.

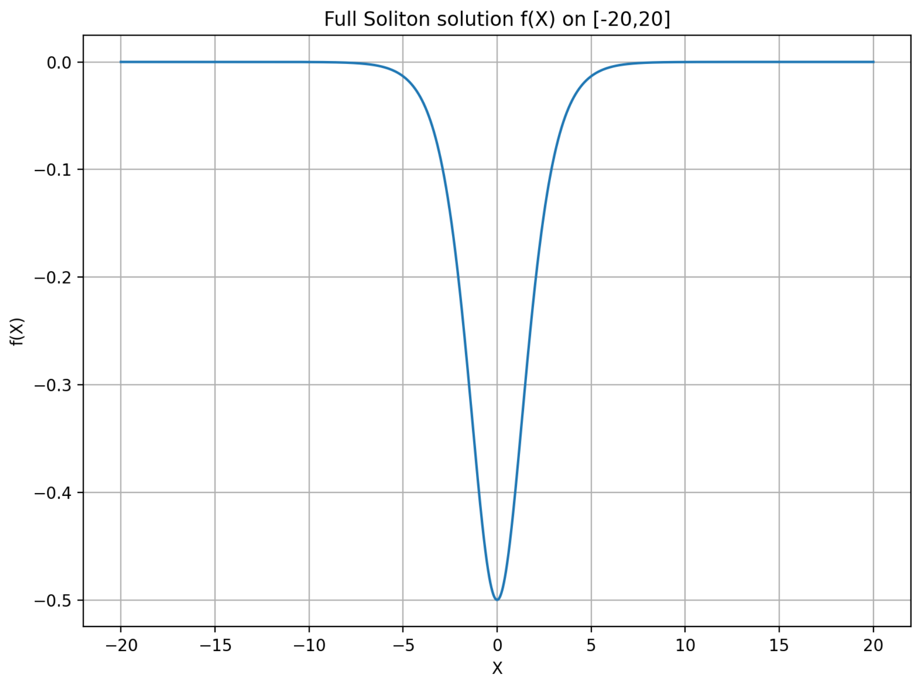

Figure 2: Full Soliton solution $f(X)$ on $[-20, 20]$

4.2 $C(X)$ and error in the Constant of Motion

Next, to test the accuracy of these solutions, plots of the constant of motion $C(X)$, alongside its absolute error $\delta C(X)$ were made (Figures 3 and 4 respectively).

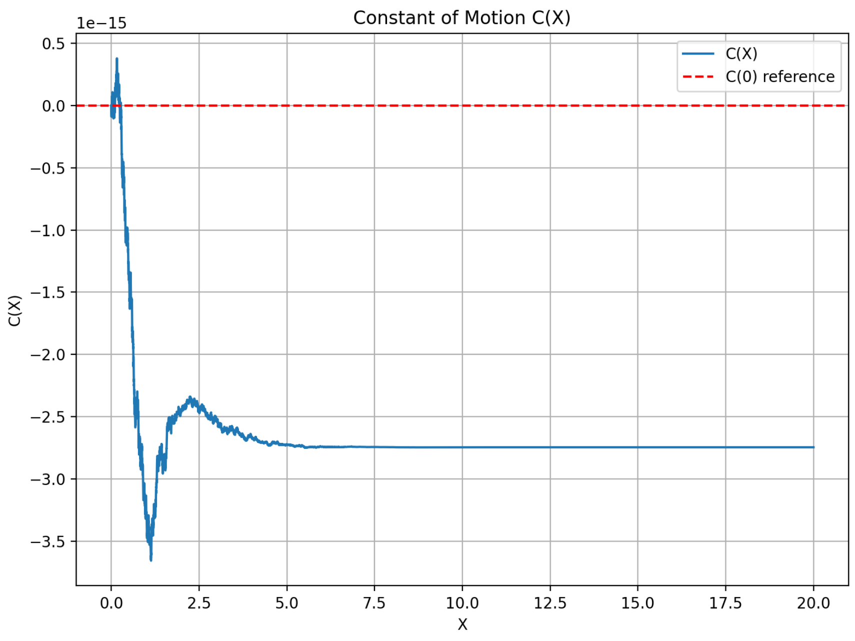

Figure 3: Constant of Motion $C(X)$ versus X

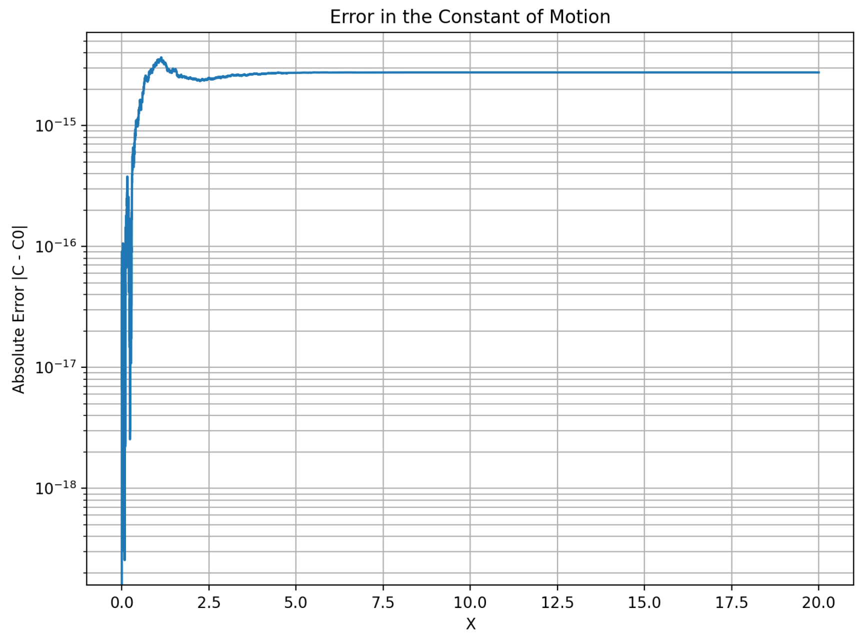

Figure 4: Absolute Error in the Constant of Motion $\delta C(X) = |C(X) - C(0)| = |C(X) - 0| = |C(X)|$

As we can see, both plots confirm that our results were extremely accurate. The changes of $C(X)$ are of the order 10^{-15}, which is the accuracy that was used in writing the data. In fact, we actually start approaching machine precision due to how accurate these solutions are, which means that if we were to decrease the step size (i.e., increase the number of intervals), the errors would actually start increasing due to the precision of the C/C++ programming language.
Now, having verified the low error in our $C(X)$ values, we are ready to begin constructing the full traveling wave solutions of our solitons.

5 Traveling Wave Reconstruction

Recall that from our traveling wave ansatz, we had

$$\phi(x,t) = f(x-t) \tag{31}$$

where we have set $c = 1$. Physically, as we shall soon see, this expression is describing a waveform that propagates to the right without changing its shape, which is the defining property of a soliton.
The numerical solution for $f(X)$ is only known at discrete grid points $X_n = nh$, where h is the step size and n is the number of steps taken. However, finding $\phi(x,t)$ at fixed values of x and t generally requires values of $f(X)$ at non-grid locations $X = x - t$. Therefore, to obtain these points, a third-order Lagrange interpolation algorithm was implemented. This gives us enough accurate solutions while keeping the code computationally inexpensive. This is consistent with the overall fourth-order accuracy of the RK4 method.
Using this interpolation, the solution $\phi(x,t)$ was calculated on the interval $x \in [-10, 10]$ for the times

$$t = 0\ ,\ 1,\ 2,\ 3,\ 4,\ 5 \tag{32}$$

After having plotted these values, we obtained Figure 5, which is precisely what a soliton looks like.

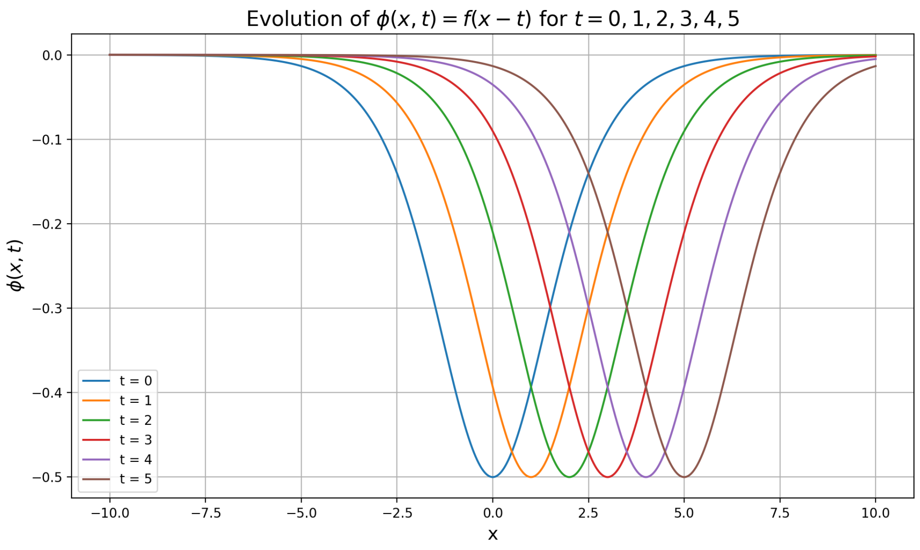

Figure 5: Time evolution of the soliton solution $\phi(x,t) = f(x-t)$ at $t = 0, 1, \ldots, 5$.

As expected for a soliton, the wave is moving uniformly in the positive x-direction while keeping its shape and amplitude. No change in the shape of the wave is visible as time increases. This is great, as it shows us that the numerical solution was able to successfully recreate the physics of a soliton that was modeled by the KdV equation.

6 Convergence of the Constant of Motion

To test the accuracy of our numerical solutions, we need to test the convergence of our solutions. This means that as we decrease the step size h (increase the number of intervals N), we must have it so that $\delta C(X)$ converges to 0. We shall test this for $\delta C(X = 5)$ now.
To do this, the value of

$$\delta C(h) = |C(X = 5; h)| \tag{33}$$

was calculated for a sequence of decreasing step sizes h. Since $C(0) = 0$, an absolute error was used rather than a relative error. For each step size $h = (X_{\text{final}} - X_{\text{initial}})/N$, the system was integrated from $X = 0$ to $X = 5$ using the RK4 method. The value of $C(X)$ at $X = 5$ was calculated, and its value was stored. The number of intervals used were $50, 100, 200, 400, 800, 1600, 3200, 6400$, which correspond to halving the step size at each run.
Figure 6 shows a log-log plot of δC versus h.

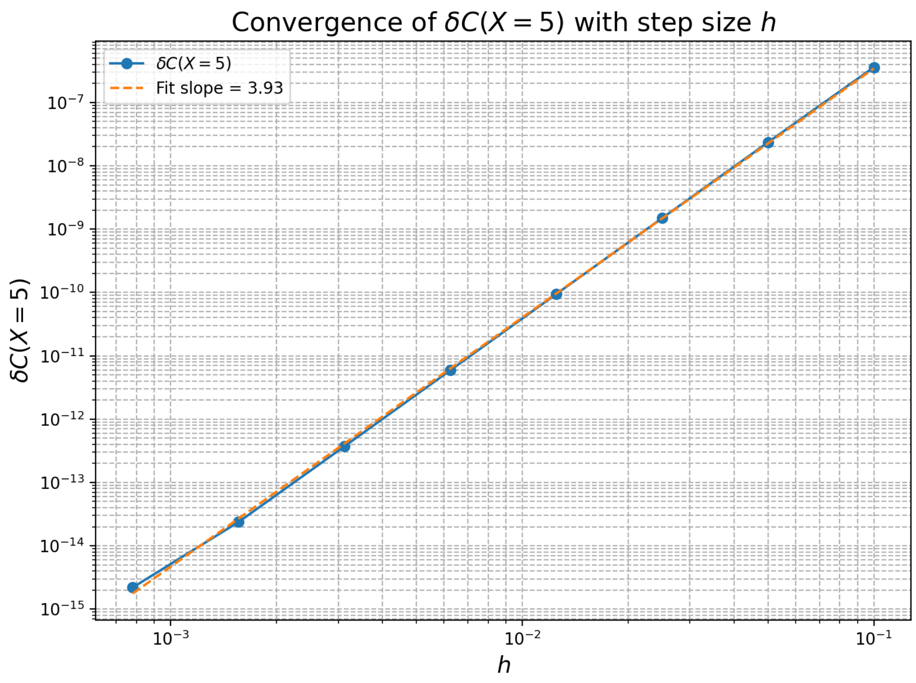

Figure 6: Log-Log plot of the error in the constant of motion at $X = 5$ as a function of step size h

From this figure, we can see that the error in the constant of motion decreases rapidly as the step size is reduced. By fitting a polynomial onto this log-log data using the *Numpy* library, we can see that a slope of 3.93 was found. This implies that

$$\delta C \propto h^{3.93} \tag{34}$$

which is close to the

$$\delta C \propto h^{4} \tag{35}$$

that we would expect from the global truncation error of the fourth-order Runge-Kutta method.
This confirms that the numerical integration is converging at the correct order and that the constant of motion is being preserved to a high accuracy as the step size decreases.
If we were to increase the number of intervals N any further, we would actually start reaching the limits of double-precision floating-point accuracy of the computer. Overall, this convergence test was very successful in showing that the numerical integration converges at the correct order and that the constant of motion is being preserved to a very high accuracy.

7 Self-Convergence of Solutions

Since the analytical solution $f(X)$ is not explicitly known for this equation, a self-convergence test is also necessary to further validate the numerical method. This method compares the numerical solutions obtained at different resolutions to test convergence without needing an exact reference solution.
Let $f(h)$ be the numerical value of $f(X = 5)$ at step size h, and let h_2 and h_3 be the two finest resolutions (N being the largest for these two cases). Then similar to how the process was performed in homework 6, we can form the ratio,

$$R_{\text{num}}(h) = \frac{f(h) - f(h_3)}{f(h_2) - f(h_3)} \tag{36}$$

as well as the expected convergence ratio,

$$R_{\text{expected}} = \frac{(h/h_3)^n - 1}{2^n - 1} \tag{37}$$

where for the RK4 scheme, $n = 4$. By plotting these two ratios on the same plot, we get Figure 7.

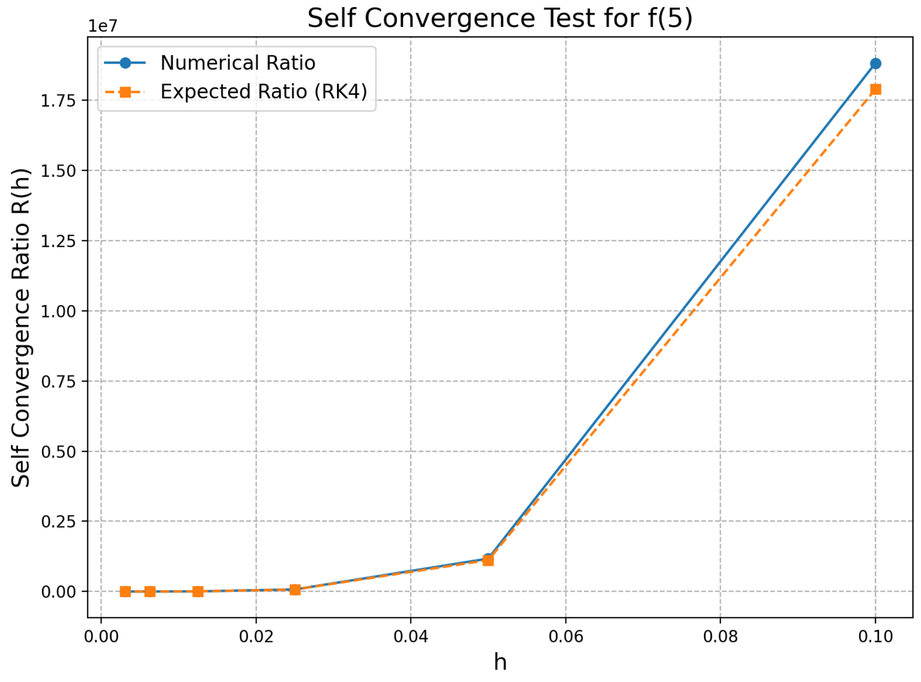

Figure 7: Self-Convergence Test for $f(X = 5)$

From the figure, we can see an excellent agreement between the numerical and expected convergent ratios. As h is large, meaning that the number of intervals N is small, the two ratios deviate from one another. However, as we refine our intervals and increase the number of steps, they get closer and closer to one another, getting to a point where they become nearly indistinguishable from each other.
Similar to the previous parts, if we were to increase the number of steps much more than the current values used, the solutions would become so accurate that the round-off error would become comparable to or larger than the truncation error.
Overall, this self-convergence test also confirms that our solutions are self-converging for fixed initial conditions at $X = 5$. Together with the convergence test of the constant of motion from Section 6, this gives us very strong evidence that the numerical implementation is correct and it behaves as theoretically expected.

8 Richardson Extrapolation

Having shown that the numerical solutions converge properly, we can now use the Richardson extrapolation technique to estimate the numerical error of solutions. This method uses the known convergence order, we can create higher-accuracy estimates from two solutions that were calculated at different step sizes.
Let f_{h_2} and f_{h_3} be the numerical solutions for $f(X = 5)$ calculated using step sizes h_2 and $h_3 = h_2/2$, which are 6400 and 3200 here. Since the RK4 method is fourth order, the Richardson-extrapolated value is given by

$$f_R = \frac{2^n f_{h_3} - f_{h_2}}{2^4 - 1} = \frac{2^4 f_{h_3} - f_{h_2}}{2^4 - 1}. \tag{38}$$

Using the numerical values from the program:

$$f_{h_2} \approx -1.329611334171813 \times 10^{-2} \qquad f_{h_3} \approx -1.329611334159839 \times 10^{-2} \tag{39}$$

Substituting these values into the Richardson extrapolation formula gives

$$f_R \approx -1.329611334159040 \times 10^{-2}. \tag{40}$$

We can now estimate the numerical error in our highest resolution solution by taking the difference between the Richardson-extrapolated value f_R and the finest-grid result (highest number of intervals N):

$$\text{estimated error} = |f_R - f_{h_3}| \approx 7.98 \times 10^{-15} \tag{41}$$

Similar to the previous sections, this error is again on the order of the double-precision floating-point round-off, showing that the solution at $N = 6400$ is very close to reaching machine precision. As a result, smaller step sizes would not significantly improve the accuracy of the solutions and would instead increase the error due to round-off error becoming comparable with the truncation error.
The Richardson extrapolation results once again confirm that the RK4 solution for $f(X = 5)$ is highly accurate and consistent with the convergence behavior from the previous two sections.

9 Conclusions

In this project, we indeed showed that numerical methods can be used to study soliton solutions of the KdV equations. By reducing the problem to a traveling wave formulation, the important features of the solution were studied without having to solve the full time-dependent PDE.
The numerical results show that the solutions were very accurate. The constant of motion is preserved to almost machine-level precision, and the 4th-order convergence confirms that the numerical method behaves exactly as expected. Self-convergence tests and Richardson extrapolation further show the high levels of accuracy in this numerical scheme.
Most importantly, the main goal of this project, that being the reconstruction of the traveling wave solutions, clearly shows the behaviors that we expect from a soliton: localization and shape preservation of the wave as it travels through space. Together, these results show that the numerical implementations were not only mathematically accurate but also physically reasonable. Overall, this project was just a small showcase of how computational physics can allow us to study complicated systems that are difficult or even impossible to solve analytically and why numerical methods are such powerful tools at the hands of physicists and researchers.

References

[1] Arshia Akhtarkavan. Solitondynamics. https://github.com/AaKavan/SolitonDynamics, 2025. GitHub repository, accessed December 2025.

[2] Philip G Drazin and Robin Stanley Johnson. *Solitons: an introduction*, volume 2. Cambridge university press, 1989.