NUMERICAL SIMULATIONS OF THE STOCHASTIC KDV EQUATION

Andrew Rose

A Thesis Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
Of the Requirements for the Degree of
Master of Science

Department of Mathematics and Statistics

University of North Carolina Wilmington

2006

This thesis has been prepared in the style and format

Consistent with the journal

American Mathematical Monthly.

TABLE OF CONTENTS

ABSTRACT

We study the Korteweg-de Vries (KdV) equation with external noise and compare our numerical simulations to known theoretical results. By using a modification of the Zabusky-Kruskal finite difference scheme, we are able to generate numerical solutions to the stochastic KdV. We look at the large time behavior of the stochastic KdV and verify the diffusion of solitons. We find that the predicted large time behavior of the perturbed soliton is not easily confirmed in the simulations as the initial soliton diffuses and is lost amidst the background noise long before the asymptotic limit is reached.

DEDICATION

For my parents and my loving wife, who have supported me unconditionally not just during the course of writing this thesis, but throughout my whole school career.

ACKNOWLEDGMENTS

I would especially like to acknowledge the enthusiastic supervision of Dr. Russell Herman, who has guided me from the first formulation to the last writing of this thesis. I would also like to acknowledge Dr. Gabriel Lugo and Dr. David Rolls, who have helped build the foundation for which my thesis is based. I would like to mention Dr. Thad Dankel, Jr., whose research has contributed to the literature on the Stochastic KdV equation and who passed away while I was completing my research. Finally, I would like to acknowledge my wife Kristine Rose and my family, for without their support I would have given up a long time ago.

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

The Korteweg-de Vries (KdV) equation, given by

$$u_t + 6uu_x + u_{xxx} = 0 \tag{1}$$

describes the generic evolution of long shallow waves with quadratic nonlinearity and third order dispersion [1]. The KdV equation occurs in many fields of physics such as in water waves, plasmas, and fiber optics. Since its discovery 111 years ago (1895), researchers have investigated the solitary wave solutions to Equation (1), including special solutions called *solitons*.

In this thesis we will look at the effects of external noise on the soliton solution of the KdV equation modeled by the stochastic KdV equation

$$u_t + 6uu_x + u_{xxx} = \zeta(t), \tag{2}$$

where the inhomogeneous term $\zeta(t)$ represents one type of external noise. Miki Wadati studied this equation analytically [9] and he determined the large time behavior of one soliton solutions under this type of noise term. In doing this he discovered the process now called “the diffusion of a soliton”.

The KdV soliton can be simulated using the Zabusky-Kruskal scheme, which will be explained in detail later in the paper. We have simulated solutions of Equation (1) and compared our results with Wadati's theoretical results. We use a modification of the Zabusky-Kruskal scheme with the added noise term to simulate our solutions.

We also have looked at the equation

$$u_t + 6uu_x + u_{xxx} - \gamma u = \zeta(t),\ \gamma > 0. \tag{3}$$

which includes a damping term (γu) added to the stochastic KdV. Equation (3) can also be simulated using the Zabusky-Kruskal scheme with minimum error by adding a correction term. Wadati and Akutsu have studied Equation (3) extensively and we compared our results to their theoretical results [10] as well.

Finally, we have looked at the two soliton solution [6], which can be written as

$$u(x,t) = \frac{2(p^2-q^2)(p^2+q^2\operatorname{sech}^2\chi(x,t)\sinh^2\theta(x,t))}{(p\cosh\theta(x,t)-q\tanh\chi(x,t)\sinh\theta(x,t))^2} \tag{4}$$

where the phases are given by

$$\theta(x,t) = px - 4p^3(t-t_0) \tag{5}$$

and

$$\chi(x,t) = qx - 4q^3(t-t_0). \tag{6}$$

We have found that the two soliton solution behaves exactly as predicted in [6], with the soliton interaction not changing the identity of each individual soliton, but only changing the soliton positions by a constant shift t. Our preliminary studies show that noise and damping affect the two soliton solution in the same way that they affect the one soliton solution, leading to the dispersion of the solitons.

In Chapter 2 we review the history of the one soliton solution beginning with the first observation of John Scott Russell. Such solitary waves show a balance between nonlinearity and dispersion. We will show how one can obtain the correct form for the solution to Equation (1). We then review in Chapter 3 the theoretical analysis of the stochastic KdV and damped stochastic KdV. These studies by Wadati [9] and Wadati and Akutsu [10] predict that the soliton given by the averaged stochastic KdV equation will have diffusion that is clearly seen with an amplitude decay on the order of $t^{-3/2}$ ($t \to \infty$) and that the soliton given by the averaged damped stochastic KdV equation will have other factors besides diffusion affecting it's height and width; it will also have an amplitude decay on the order of $t^{-1/2}\,e^{-\gamma t}$ $(t \to \infty)$.

To test these theories we have first numerically simulated the statistical averages of the exact solution and Wadati's theoretical average. Then we numerically solved the stochastic KdV and damped stochastic KdV equation using a well-known finite difference scheme combined with numerical realizations of Gaussian white noise. In Chapter 4 we explain the Zabusky-Kruskal finite difference scheme for the KdV and in Chapter 5 we discuss the numerical realizations of Brownian motion and Gaussian white noise. We combine these techniques in Chapter 6 and explain our choices in the way we have written the numerical code. We explain certain nuances of MATLAB that have required us to make use of its inherent vectorization technique and prove that our code produces results akin to the predictions mentioned previously.

Finally, we present the results of the simulations in Chapter 7. We have found that we can closely reproduce numerically Wadati's prediction for an amplitude decay of $t^{-3/2}$ for the one soliton solution correctly (with a small amount of error due to our numerical scheme), and that the effect of diffusion is clearly seen on the soliton's amplitude. However, we were not able to come to a clear conclusion on Wadati's and Akutsu's prediction for the amplitude decay of the damped stochastic KdV with our numerical simulations. The amplitude decay of $t^{-1/2}\,e^{-\gamma t}$ was seen in some cases, but we had to use a larger time interval; this kept us from being able to estimate the decay order because of background noise. Other types of approximations can be tried in the future to see if it is possible to track the soliton through this background noise. For the two soliton solution, we were able to verify that two solitons do retain their identity after interaction when using the Zabusky-Kruskal scheme. We were also able to see the effects of noise and damping on the individual solitons making up the two soliton solution.

In the last chapter we summarize our results, compare them to known results produced by previous studies, and indicate a possible direction for future studies. In the Appendix we show the code used to produce all of the results in this paper. It is possible to expand the code and use it for later research on the effects of other types of noise and perturbations for N-soliton solutions.

2 THE KDV SOLITON

One of the most interesting wave phenomena that exists in the natural world is that of solitary waves. A simple definition for a solitary water wave is "a wave that consists of a single elevation (a rounded, smooth, and well-defined heap of water), neither proceeded nor followed by another elevation (or depression)" [1]. This definition is only for one instance of solitary waves; they also exist in other physical mediums. Although some waves can be modeled by simple expressions relating the amplitude, frequency, and speed to the medium through which they propagate, solitary waves can have a much more complex dependance on both space and time.

Solitary waves were discovered in 1834, when John Scott Russell (who was a Scottish civil engineer) witnessed horses pulling a barge in the Union canal in Edinburgh. He later wrote that when he saw that the barge came to a stop, a wave "rolled forward with great velocity assuming the form of a large solitary elevation, a rounded heap of water, which continued its course without change of form or diminution of speed." Russell kept up with it for as long as possible until the wave finally dissipated. Afterwards, Russell created comparable waves in his basement laboratory by lowering weights into a rectangular tank full of water. By doing this, Russell confirmed the existence of solitary waves and learned that the wave's speed depends on its amplitude and the depth of the water. The solitary waves moved faster in deeper water, with the taller waves traveling faster than the flatter waves.

Unfortunately, most scientists (including George Airy, who did not agree with the emphasis John Scott Russell placed on solitary waves) did not believe this type of wave really existed until 1870 (although in 1849 George Stokes, an Irish physicist, showed that solitary waves could indeed arise from a combination of periodic waves) [2]. In 1870, English physicist John Rayleigh and the French mathematician Joseph Boussinesq independently proved that these waves existed by using the basic equations of fluid dynamics. But it wasn't until 1895 that Diederik Johannes Korteweg and Gustave de Vries derived an equation that described the actual propagation of the waves that Russell first witnessed and proved Rayleigh and Boussinesq's idea of stability. Thus, the question of whether the equations for water waves allowed the existence of solitary wave solutions was finally answered sixty years after Russell's first observation in the canal [2].

The Korteweg-de Vries equation (or KdV equation as it will henceforth be called)

$$u_t + 6uu_x + u_{xxx} = 0. \tag{7}$$

describes the evolution of long waves (with large length and measurable amplitude) down a canal with a rectangular cross section.

Here u represents the wave amplitude, and u_t and u_x are the partial derivatives with respect to t and x, respectively. The quantity u_t represents the vertical velocity of the wave at (x, t), u_x describes the rate of change in amplitude with respect to x, and u_{xxx} is a dispersion term. This means that if u is the amplitude of a wave at some point in space, then u_x is the slope of the wave at that point and u_{xx} is the concavity near the point [2].

The existence of solitary waves is due to the balancing effects of uu_x and u_{xxx} in Equation (7). The nonlinear term $6uu_x$ in Equation (7) is important because the amplitude of the wave depends on it's own rate of change in space; it also represents steepening. The term u_{xxx} implies dispersion of different frequency components. For linear problems, dispersive waves usually are characterized by solutions

$$\varphi(x, t) = Ae^{i\kappa x - i\omega t}, \tag{8}$$

where κ is the wave number, ω is the frequency, and A is the amplitude. In fact, the dispersion relation, written as $\omega = \omega(\kappa)$, coupled together with the nonlinear term mentioned previously, is what produces the balance between nonlinearity and dispersion and generates solitary waves (instead of the formation of other known waves) [11].

In 1955 at the Los Alamos Scientific Laboratory, Enrico Fermi, John Pasta and Stanislaw Ulam used Maniac I (a very powerful computer at the time) to work on a problem involving the conduction of heat in solids. They modeled a solid as a one-dimensional network of masses connected by springs, with heat in the system embodied in the vibrational motion of the masses. The simplest version of the model employs a linear equation to describe the action of the springs; Fermi and his collaborators specified a nonlinear law: the force exerted by a spring would be given by the sum of the displacement and a small term proportional to the square of the displacement. They expected energy to diffuse evenly through the network. Instead, they found that if a pulse of energy was applied to one end of the network, it flowed through all nodes and it's position took on a random distribution. Finally, in time the pulse reassembled and returned to it's initial state [3].

Norman J. Zabusky and Martin D. Kruskal were working on analyzing this mass-and spring model in 1965 by considering what happens when the length of the springs tend toward zero when they made the discovery that the limiting equation defining the continuous system was the KdV equation. Later, Zabusky and Kruskal explored the KdV equation through extensive computer simulations. They found that when examining multiple interacting solitary waves, the waves collided and separated again with their size, shape, and speed unchanged. The only difference was that the colliding waves had shifted from the position they would have had if there was no collision. Hence they gave these special waves the name of *solitons*, because these waves acted like particles of light [3].

A simple analytical form for the KdV soliton is given by the following expression:

$$u(x,t) = 2\eta^2 \operatorname{sech}^2 \eta(x - x_0 - 4\eta^2 t). \tag{9}$$

This soliton is centered at $x_0 + 4\eta^2 t$, has an amplitude of $2\eta^2$, a width of $\frac{1}{\eta}$, and an initial position at x_0. Thus, it is a traveling wave of constant amplitude and width moving at speed $4\eta^2$.

This solution is easily obtained from the KdV equation. We assume in Equation (7) that $u(x,t)$ is a traveling wave solution of the form

$$u(x,t) = f(x - ct) = f(\xi), \tag{10}$$

where c is the constant speed of the traveling wave [1]. Substituting Equation (10) into Equation (7) gives us the ordinary differential equation

$$-c\frac{df}{d\xi} + 6f\frac{df}{d\xi} + \frac{d^3 f}{d\xi^3} = 0. \tag{11}$$

This in turn can be written as a perfect derivative,

$$\frac{d}{d\xi}\left[-cf + 3f^2 + \frac{d^2 f}{d\xi^2}\right] = 0. \tag{12}$$

Equation (12) can be integrated to give

$$-cf + 3f^2 + \frac{d^2 f}{d\xi^2} = A, \tag{13}$$

where A is an arbitrary constant. Multiplying Equation (13) by $\frac{df}{d\xi}$ yields

$$-cf\frac{df}{d\xi} + 3f^2\frac{df}{d\xi} + \frac{d^2 f}{d\xi^2}\frac{df}{d\xi} = A\frac{df}{d\xi}. \tag{14}$$

This can be rewritten as an exact derivative:

$$\frac{d}{d\xi}\left[\frac{1}{2}\left(\frac{df}{d\xi}\right)^2 + f^3 - \frac{1}{2}cf^2 - Af\right] = 0. \tag{15}$$

Integrating, we find the first order differential equation

$$\frac{1}{2}\left(\frac{df}{d\xi}\right)^2 = -f^3 + \frac{1}{2}cf^2 + Af + B. \tag{16}$$

We seek solutions such that $u \to 0$, $u_x \to 0$, and $u_{xxx} \to 0$ as $|x| \to \infty$ (otherwise, we get periodic traveling wave solutions which are called cnoidal waves). Thus, from Equations (13) and (16) we set $A = 0$ and $B = 0$.

Equation (16) can now be rewritten as

$$\left(\frac{df}{d\xi}\right)^2 = f^2(c - 2f). \tag{17}$$

This can be solved by using the method of separation of variables. We obtain

$$\int \frac{df}{f\sqrt{c-2f}} = \int d\xi. \tag{18}$$

The integration of the left side of Equation (18) can be done using the transformation

$$f = \frac{1}{2} c \operatorname{sech}^2\theta. \tag{19}$$

This gives us

$$c - 2f = c(1 - \operatorname{sech}^2\theta) = c\tanh^2\theta \tag{20}$$

and

$$df = -c\operatorname{sech}^2\theta \tanh\theta \, d\theta. \tag{21}$$

Substituting Equations (19)-(21) into Equation (18) gives us

$$\xi - \xi_0 = -\frac{2}{\sqrt{c}} \int \frac{1}{\operatorname{sech}^2\theta \tanh\theta} \frac{\sinh\theta}{\cosh^3\theta} d\theta, \tag{22}$$

which simplifies to

$$\xi - \xi_0 = -\frac{2}{\sqrt{c}} \int d\theta \tag{23}$$

or

$$\theta = -\frac{\sqrt{c}}{2}(\xi - \xi_0). \tag{24}$$

Now, we can finally substitute θ into Equation (19) which yields

$$f(\xi) = \frac{c}{2}\operatorname{sech}^2(\frac{\sqrt{c}}{2}(\xi - \xi_0)). \tag{25}$$

Since $u(x,t) = f(x - ct)$, we have

$$u(x,t) = \frac{c}{2}\operatorname{sech}^2\left[\frac{\sqrt{c}}{2}(x - ct - \xi_0)\right]. \tag{26}$$

Writing the wave speed as $c = 4\eta^2$ and the initial position $\xi_0 = x_0$, Equation (26) becomes

$$u(x,t) = 2\eta^2 \operatorname{sech}^2 \eta(x - x_0 - 4\eta^2 t), \tag{27}$$

which is the one soliton solution of KdV as indicated in Equation (9).

Figure 1 shows an example of this solution on the region $[-10, 40] \times [0, 4]$. In this figure we can see that the wave moves across the interval unchanged.

The interesting discovery in Zabusky and Kruskal's work with the KdV equation is the behavior of N-Soliton solutions. When two solitons collide, they interact elastically. The exact solution for the two soliton equation is given by [6]

$$u(x,t) = \frac{2(p^2 - q^2)(p^2 + q^2 \operatorname{sech}^2 \chi(x,t) \sinh^2 \theta(x,t))}{(p \cosh \theta(x,t) - q \tanh \chi(x,t) \sinh \theta(x,t))^2} \tag{28}$$

where the phases are

$$\theta(x,t) = px - 4p^3(t - t_0) \tag{29}$$

and

$$\chi(x,t) = qx - 4q^3(t - t_0). \tag{30}$$

In our simulations we take $p = 2$, $q = 1.5$ and $t_0 = 0.5$. Here p and q can be varied to produce solitons of differing amplitudes and widths. The two solitons have amplitudes of

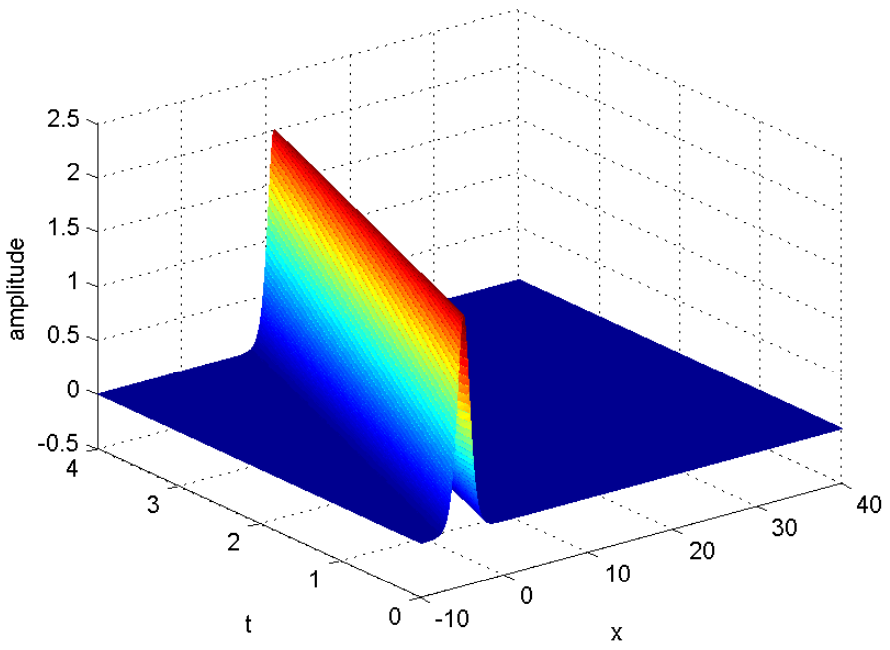


Figure 1: The one soliton solution with $\eta = 1$ and $x_0 = 5.0$.

$2p^2$ and $2q^2$, and are traveling at a speed of $4p^2$ and $4q^2$ respectively. This means that we can pick our p and q according to which wave we want to be larger (and henceforth have a faster speed).

Figure 2 shows an example of the two soliton solution on the domain $[-10, 10] \times [0, 1]$ with $p = 2$, $q = 1.5$, and $t_0 = 0.5$. As expected, we see that the two solitons travel without change

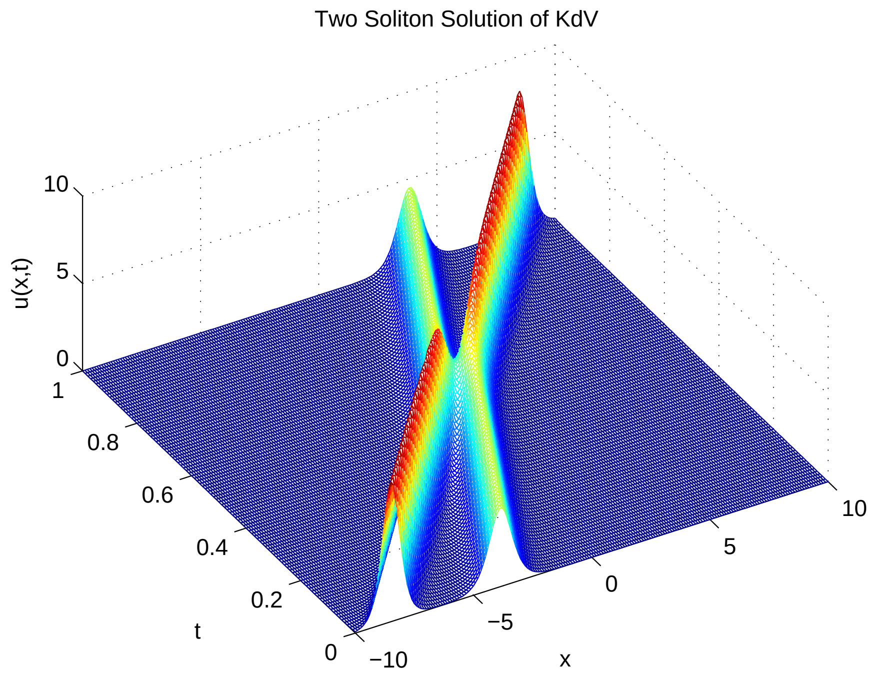


Figure 2: 3D plot of the two soliton solution.

until they collide; but afterwards, while their amplitudes are unchanged, their position in time has changed considerably.

Figure 3 shows a 2-d plot of the interaction between two solitons on the domain $[-10, 10] \times [0, 1]$ with $p = 2$, $q = 1.5$ and $t_0 = 0.5$. The two solitons initially approach each other with unchanging amplitude. When they collide, the two solitons do not merge into a single peak. (The only way for this to happen is if $p/q > \sqrt{3}$) [6]. However, once the solitons re-emerge, their amplitudes are the same but they experience a phase shift [6]. In general, the taller soliton is shifted by $\frac{4}{p}(\ln \frac{p+q}{p-q})$ and the shorter soliton is shifted by $\frac{4}{q}(\ln \frac{p+q}{p-q})$.

Later analytical work by Clifford S. Gardner, John M. Greene, Martin Kruskal, and

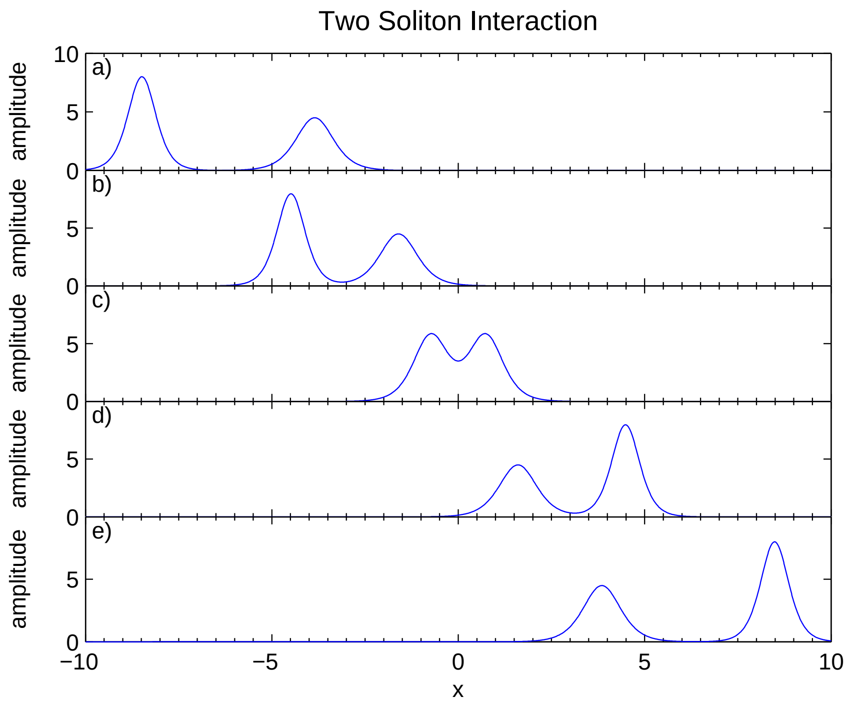


Figure 3: 2D plot of two solitons interacting over the time interval $[0, 1]$.

Robert M. Miura of Princeton University on the initial value problem for the KdV led to the search for N-soliton solutions in other systems, paving the way for most of the soliton research being done today [2].

Since the first discovery of solitary waves in 1834, a great deal of research has been done on their formation and subsequent evolution. In spite of initial successes in this research, to this day the KdV equation can only be solved in closed form in special cases. Also, occurrences of these solitary waves in nature are usually subjected to additional forces, leading to perturbations of the equation. For example, external noise (like ridges on the floor of a long narrow canal) and friction (like damping) can affect the way the wave behaves.

To study the evolution of solitons under perturbations, we are often forced to do numerical simulations. The simplest finite different scheme for the KdV is the one used by Zabusky-Kruskal and will be described in Chapter 4. However, we will first consider the stochastic KdV equation.

3 THE STOCHASTIC KDV

As is the case with most natural phenomena, solitary waves may be subject to random perturbations. In this chapter we consider adding a Gaussian noise term $\zeta(t)$ to Equation (1):

$$u_t + 6uu_x + u_{xxx} = \zeta(t). \tag{31}$$

Gaussian white noise $\zeta(t)$ has zero mean and is δ-correlated [9]:

$$< \zeta(t)\zeta(t') >= 2\epsilon\delta(t-t'). \tag{32}$$

The solution to Equation (31) is now considered to be a random variable. Here, ϵ is a small number and $\delta(t)$ is the Dirac delta function. Equation (31) is called a stochastic partial differential equation, with u now depending on a random noise term. We will follow Wadati [9], with a difference in sign in the nonlinear term of the KdV; the results will essentially be the same because physically a depression for $-6uu_x$ exists as opposed to a hump for $+6uu_x$.

Wadati [9] has shown that for time-dependent noise, the stochastic KdV equation can be transformed into an unperturbed KdV equation (as seen in Equation (7)),

$$U_T + 6UU_X + U_{XXX} = 0, \tag{33}$$

by introducing the Galilean transformation (which is a transformation between two coordinate systems in constant relative motion)

$$\begin{aligned} u(x,t) &= U(X,T) + W(T), && \text{(34a)} \\ X &= x + m(t), && \text{(34b)} \\ T &= t, && \text{(34c)} \\ m(t) &= -6\int_0^t W(t')\,dt'. && \text{(34d)} \end{aligned}$$

This can be seen as follows. From the chain rule, we have for a composite function of X and T:

$$\begin{aligned}\frac{\partial}{\partial x} &= \frac{\partial X}{\partial x}\frac{\partial}{\partial X} + \frac{\partial T}{\partial x}\frac{\partial}{\partial T} \\ &= \frac{\partial}{\partial X}, \end{aligned} \tag{35}$$

and

$$\begin{aligned}\frac{\partial}{\partial t} &= \frac{\partial X}{\partial t}\frac{\partial}{\partial X} + \frac{\partial T}{\partial t}\frac{\partial}{\partial T} \\ &= -6W(T)\frac{\partial}{\partial X} + \frac{\partial}{\partial T}. \end{aligned} \tag{36}$$

We apply the transformations in Equations (34a)-(34d) to Equation (31) and find

$$\begin{aligned}\zeta(t) &= u_t + 6uu_x + u_{xxx}, \\ &= (U+W)_T - 6WU_x + 6(U+W)U_X + U_{XXX}, \\ &= U_T + 6UU_X + U_{XXX} + W_T. \end{aligned} \tag{37}$$

Defining

$$\zeta(t) = W_T,$$

or

$$W(T) = \int_0^T \zeta(t')\,dt', \tag{38}$$

Equation (37) becomes the unperturbed KdV equation in (33). Therefore, we obtain the result

$$u(x,t) = U(x+m(t),t) + W(t). \tag{39}$$

This is true for any exact solution of the KdV equation U.

Before we go further, we will discuss Brownian motion, from which we derive Gaussian white noise, and the definition of stochastic processes. This is necessary to understand what role Brownian motion plays in the formulation of the stochastic KdV equation.

Brownian motion was first discovered by Robert Brown in 1827, while studying pollen particles floating in water under a microscope [7]. He noticed that the small particles moved in a random fashion and he became intrigued by the cause of this motion. Brown was able to rule out the possibility of the pollen being alive by using particles of dust but an explanation of the random motion was not given until Louis Bachelier wrote his 1900 Ph.D. thesis, "The Theory of Speculation." In 1910 Albert Einstein and Marian Smoluchowski further cemented the explanation when they described how the particles collided with randomly moving molecules of water. Small particles would receive a random number of impacts of random strength from random directions over a period of time [7]. This work was based on of one of Einstein's famous 1905 papers for which the world celebrated the centennial this past year (2005).

In a series of papers originating in 1918 [7], Wiener formalized the theory of Brownian motion in terms of stochastic processes. To understand stochastic processes, it is necessary to introduce random variables. A formal definition of random variables can be found in [7] but for the purposes of this work it suffices to think of a random variable as a variable whose value is not known but whose statistical distribution is. Now we are able to define stochastic processes:

Definition 1 *A* ***Stochastic Process*** $(X(t), t \in T)$ *is a collection of random variables. That is for each* $t \in T$, $X(t)$ *is a random variable.*

Because the index t is often interpreted as time, we can refer to $X(t)$ as the state of the process at time t. The set T is called the index set of the process. However, in the case of Brownian motion, T defines an interval of the real line, $0 \leq t \leq T$ because its increments are independent and stationary.

Brownian motion is considered a continuous-time stochastic process [7]. A standard

Brownian motion, or standard Wiener process, over $[0, T]$ is a collection of random variables $W(t)$ that depend continuously on $t \in [0, T]$. Furthermore, Brownian motion satisfies the following three conditions [5]:

1. $W(0) = 0$ (with probability 1).
2. For $0 \leq s < t \leq T$ the random variable given by the increment $W(t) - W(s)$ is normally distributed with mean zero and variance $t - s$; equivalently, $W(t) - W(s) \sim \sqrt{t-s}N(0, 1)$, where $N(0, 1)$ denotes the standard normal distribution.
3. For $0 \leq s < t < u < v \leq T$ the increments $W(t) - W(s)$ and $W(v) - W(u)$ are independent, which means that each increment does not depend on any of the preceding increments.

We also note that $W(t)$ has a probability density function given by:

$$f_t(x) = \frac{1}{\sqrt{2\pi t}} \exp^{-x^2/2t}, \quad -\infty < t < \infty. \tag{40}$$

Thus, averages (or expectation values) of random variables may be computed as integrals of the form $< g(x) >= \int_{-\infty}^{\infty} g(x) f(x)\, dx$. Therefore, we can easily compute averages of solutions of the stochastic KdV equation.

We consider the one-soliton solution of Equation (33) in the form

$$U(X, T) = 2\eta^2 \operatorname{sech}^2(\eta(X - 4\eta^2 T - X_0)), \tag{41}$$

where X_0 is the new initial position. Then, the transformation in Equation (34) leads directly to an exact solution of the stochastic KdV equation (31):

$$u(x, t) = 2\eta^2 \operatorname{sech}^2 \left(\eta \left(x - 4\eta^2 t - x_0 - 6 \int_0^t W(t')\, dt' \right) \right) + W(t), \tag{42}$$

where $W(t)$ is Brownian motion (as seen in Equation (38)) and $x_0 = X_0$.

We now consider the effect of noise on the soliton's amplitude for an ensemble of solutions. For example, we can examine the behavior of the average $< u(x,t) >$ of the solution and compare it to the exact solution [9]:

$$< u(x,t) >= 2\eta^2 \left\langle \operatorname{sech}^2 \left(\eta \left(x - 4\eta^2 t - x_0 - 6 \int_0^t W(t')\, dt' \right) \right) \right\rangle .$$

Wadati computes this average by first converting the hyperbolic function into an exponential series,

$$\operatorname{sech}^2 z = 2 \sum_{n=1}^{\infty} (-1)^{n+1} n e^{2nz}. \tag{43}$$

Wadati then proceeds by computing

$$< u(x,t) >= 8\eta^2 \sum_{n=1}^{\infty} (-1)^{n+1} n \left\langle \exp \left[2n\eta \left(x - 4\eta^2 t - x_0 - 6 \int_0^t W(t')\, dt' \right) \right] \right\rangle .$$

To complete this computation, some useful relations are used (these depend on the properties of Gaussian white noise, $< \zeta >= 0$ and Equation (32)) [9]:

$$< W(t) >= 0, \tag{44}$$

$$< W(t_1) W(t_2) >= 2\epsilon \min(t_1, t_2), \tag{45}$$

$$< \exp\left(cW(t)\right) >= \exp\left(\frac{1}{2} c^2 < W^2(t) > \right). \tag{46}$$

We can use these identities to show that

$$\begin{aligned} \left\langle \exp\left(\pm 12 n\eta \int_0^t W(t')\, dt' \right) \right\rangle &= \exp\left(72 n^2 \eta^2 \int_0^t \int_0^t < W(t_1) W(t_2) > dt_2 dt_1 \right), \\ &= \exp(48 n^2 \eta^2 \epsilon t^3). \end{aligned} \tag{47}$$

This leads to the following expression for the average:

$$< u(x,t) >= 8\eta^2 \sum_{n=1}^{\infty} (-1)^{n+1} n e^{na+n^2 b}, \tag{48}$$

where

$$a = 2\eta(x - x_0 - 2\eta^2 t),$$

$$b = 48\eta^2 \epsilon t^3.$$

Wadati then develops expressions that give an analytical interpretation to this result and allow for explorations of this solution for large time [9]. Differentiating the series with respect to a and b leads to the partial differential equation

$$w_b = w_{aa} \tag{49}$$

for $w(a,b) =< u(x,t) >$. Furthermore, setting $b = 0$ in Equation (48) and comparing to Equation (43), we have that

$$w(a,0) = 2\eta^2 \operatorname{sech}^2 \frac{a}{2}. \tag{50}$$

Equations (49) and (50) give us the initial-boundary value problem for the heat, or diffusion, equation on the real line. The equation can be solved using the standard Fourier transform technique [9]. We define the Fourier transform

$$\hat{w}(k,b) = \int_{-\infty}^{\infty} w(a,b) e^{-iak} \, da, \tag{51}$$

and its inverse transform

$$w(a,b) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{w}(k,b) e^{iak} \, dk. \tag{52}$$

The heat equation leads to the simple initial value problem consisting of a first order ordinary differential equation

$$\hat{w}_b = -k^2\hat{w}, \tag{53}$$

with initial value

$$\begin{aligned}\hat{w}(k,0) &= 2\eta^2\int_{-\infty}^{\infty}\operatorname{sech}^2\left(\frac{a}{2}\right)e^{-iak}\,da \\ &= 8\eta^2\frac{\pi k}{\sinh \pi k}.\end{aligned} \tag{54}$$

The solution of this problem

$$\hat{w}(k,b) = 8\eta^2\frac{\pi k}{\sinh \pi k}e^{-bk^2} \tag{55}$$

and the solution $u(x,t)$ of the diffusion equation is then found from the inverse Fourier transform of Equation (55) as

$$u(x,t) = \frac{4\eta^2}{\pi}\int_{-\infty}^{\infty}\frac{\pi k}{\sinh \pi k}e^{iak-bk^2}\,dk. \tag{56}$$

One can apply the Convolution Theorem to this result by noting that $\hat{w}(k,b) = \hat{f}(k)\hat{g}(k,b)$ for

$$\hat{f}(k) = 8\eta^2\frac{\pi k}{\sinh \pi k}$$

and

$$\hat{g}(k,b) = e^{-bk^2}.$$

The inverse transforms for each of these are given by

$$f(a) = 2\eta^2\operatorname{sech}^2\frac{a}{2}$$

and

$$g(a,b) = \frac{1}{\sqrt{4\pi b}} e^{-a^2/4b}.$$

The last expression is just the statement that the Fourier transform of a Gaussian distribution (function $g(a,b)$) is also a Gaussian distribution [9]. Combining these results, we have

$$\begin{aligned} < u(x,t) > &= w(a,b) \\ &= (f * g)(a) \\ &= \int_{-\infty}^{\infty} f(s) g(a-s)\, ds \\ &= \int_{-\infty}^{\infty} \left(2\eta^2 \operatorname{sech}^2 \frac{s}{2}\right) \left(\frac{1}{\sqrt{4\pi b}} e^{-(a-s)^2/4b}\right) ds \\ &= \frac{\eta^2}{\sqrt{\pi b}} \int_{-\infty}^{\infty} e^{-(a-s)^2/4b} \operatorname{sech}^2 \frac{s}{2}\, ds. \end{aligned} \tag{57}$$

This result will be used later when we analyze our numerical results. We will refer to this average as $< u(x,t) >$.

Wadati shows that the asymptotic behavior of the mean solution $< u(x,t) >$ is given by

$$< u(x,t) > \sim \frac{\eta}{\sqrt{3\pi\epsilon}} \frac{1}{t^{3/2}} \exp\left(-\frac{(x - x_0 - 4\eta^2 t)^2}{48\epsilon t^3}\right), t \to \infty \tag{58}$$

where $f(t) \sim g(t)$ means $\frac{f(t)}{g(t)} \to 1$ as $t \to \infty$. Large times are those that have $b = 48\kappa^2\epsilon t^3 > 1$. In Figure 4 it is evident that the soliton undergoes a diffusion as a function of time exactly as predicted by Wadati [9]. In Chapter 7 we will verify that the diffusion is on the order of $t^{-3/2}$. Each wave in Figure 4 is actually the position of one wave as it propagates in time. In the next chapter, we will look at a numerical scheme for simulating the KdV solution and later we will look at the behavior of the waves as $t \to \infty$ (most people concentrate on the "tail" end of the soliton, but we will be considering the whole wave).

We are also interested in looking at the effect of friction on the soliton, by adding a

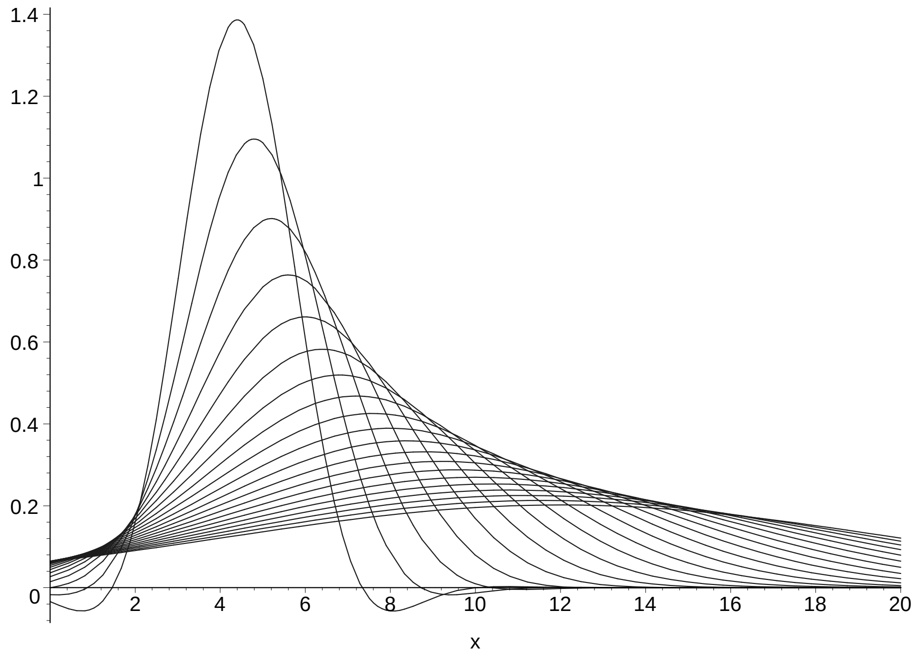


Figure 4: The exact solution based on $< u >$ in Equation (58)

damping term (γu) to the stochastic KdV equation to give

$$u_t + 6uu_x + u_{xxx} - \gamma u = \zeta(t). \tag{59}$$

Equation (59) describes the wave propagation in a one-dimensional chain. We assume that the noise $\zeta(t)$ does not depend on the coordinate x. It corresponds to the situation where the whole wave is perturbed coherently [11].

As with the stochastic KdV, we can introduce transformations of the dependent and independent variables:

$$\begin{aligned} u(x,t) &= U(X,T) + W(T), && \text{(60a)} \\ X &= x + m(t), && \text{(60b)} \end{aligned}$$

where

$$W(t) = e^{-\gamma t} \int_{t_0}^{t} \eta(s) e^{\gamma s} ds \tag{61}$$

and

$$m(t) = -6 \int_{t_0}^{t} W(s) ds. \tag{62}$$

It is readily seen that $U(x,t)$ satisfies

$$U_t + 6UU_x + U_{xxx} + \gamma U = 0, \tag{63}$$

which leads us to the non-stochastic KdV equation [11]. Approximate solutions are known for the damped KdV [10]. These will be useful in obtaining approximate solutions of a damped stochastic KdV equation.

Following a similar analysis as done with noise, we can show that [10]

$$< u(x,t) >= \frac{\eta^2}{\sqrt{\pi b}} \int_{-\infty}^{\infty} e^{-(a-s)^2/4b} \operatorname{sech}^2 \frac{s}{2}\, ds. \tag{64}$$

with

$$\begin{aligned}
\eta(t) &= \eta_0 e^{-2\gamma t}, \\
a &= 2\eta(x - x_0 - 2\eta^2 t), \\
b &= \frac{36\eta^2}{\epsilon\gamma^2}\left[2\epsilon\gamma t - 3 + 4e^{-\epsilon\gamma t} - e^{-2\epsilon\gamma t}\right] \qquad (65)
\end{aligned}$$

The large t behavior for the damped stochastic KdV is therefore predicted by Wadati to be on the order of $t^{-1/2}\, e^{-\gamma t}$. We can see this because as $t \to \infty$, the exponential part of b will disappear and we will be left with $\frac{1}{\sqrt{b}} = \frac{1}{\sqrt{t}}$. Large times are those for $b > 1$. We will explore the results of our numerical simulations in Chapter 7.

4 THE ZABUSKY-KRUSKAL SCHEME

In this chapter we consider a numerical model which can be used to simulate the solution of the KdV equation in Equation (7) for the initial condition $u(x,0) = 2\eta^2 \operatorname{sech}^2\eta(x-x_0)$. Here x_0 is the initial position of the wave. The results of the simulation are then compared to the exact solution in Equation (9). In 1965, Zabusky and Kruskal studied the KdV equation numerically using the following finite difference approximation with a centered difference in time [3]:

$$u_t = \frac{u(j,n+1) - u(j,n-1)}{2\Delta t} + O(\Delta t^2) \tag{66}$$

$$u = \frac{u(j+1,n) + u(j,n) + u(j-1,n)}{3} + O(\Delta x^2) \tag{67}$$

$$u_x = \frac{u(j+1,n) - u(j-1,n)}{2\Delta x} + O(\Delta x^2) \tag{68}$$

$$u_{xxx} = \frac{u(j+2,n) - 2u(j+1,n) + 2u(j-1,n) - u(j-2,n)}{2\Delta x^3} + O(\Delta x^2), \tag{69}$$

where $t = n\Delta t$ and $x = a + j\Delta x$. The scheme is typically used with periodic boundary conditions (ours was based on the time interval we chose).

The linear stability condition for this numerical scheme is given by [8]

$$\Delta t \leq [6\Delta x|u| + 4(\Delta x)^{-3}]^{-1}. \tag{70}$$

which can be satisfied by taking $\Delta t \leq \frac{(\Delta x)^3}{4} \leq [6\Delta x|u| + 4(\Delta x)^{-3}]^{-1}$. We now show how to derive this linear stability condition using numerical analysis.

Because this is a linear stability analysis, we use the term au_x instead of uu_x, with $a = \max_{x,t} u$ over the domain of interest. The linearized Zabusky-Kruskal scheme becomes

$$\begin{aligned} u(j,n+1) \quad = \quad & u(j,n-1) - 6a\frac{\Delta t}{\Delta x}(u(j+1,n) - u(j-1,n)) - \\ & \frac{\Delta t}{\Delta x^3}(u(j+2,n) - 2u(j+1,n) + 2u(j-1,n) - u(j-2,n)). \end{aligned} \tag{71}$$

We can seek solutions of the form $u(j,n) = \xi^n e^{ijk\Delta x}$ where ξ is the amplification factor and k is the wave number. For stability we must have $|\xi| \leq 1$. If we substitute this form of $u(j,n)$ into Equation (71) we get

$$\xi = \xi^{-1} - 6a\frac{\Delta t}{\Delta x}(2i \sin k\Delta x) - \frac{\Delta t}{\Delta x^3}(e^{2ik\Delta x} - 2e^{ik\Delta x} + 2e^{-ik\Delta x} - e^{-2ik\Delta x}), \tag{72}$$

which simplifies to

$$\xi = \xi^{-1} - 2ia\frac{\Delta t}{\Delta x}6\sin k\Delta x - 2i\frac{\Delta t}{\Delta x^3}(\sin 2k\Delta x - 2\sin k\Delta x). \tag{73}$$

Multiplying Equation (73) by ξ and using the double angle formula yields

$$0 = \xi^2 + 2i\sin(k\Delta x)\frac{\Delta t}{\Delta x}\left[6a + \frac{2}{\Delta x^2}(\cos k\Delta x - 1)\right]\xi + 1. \tag{74}$$

Defining

$$2\beta = 2i\sin(k\Delta x)\frac{\Delta t}{\Delta x}\left[6a + \frac{2}{\Delta x^2}(\cos k\Delta x - 1)\right]$$

yields

$$0 = \xi^2 + 2\beta\xi + 1$$

which leads to

$$\xi_\pm = -\beta \pm \sqrt{\beta^2 - 1} \tag{75}$$

as a solution. Note that $\xi_+\xi_- = 1$. This means that we have $|\beta| \leq 1$ as a stability condition. Now we have

$$\left|\sin k\Delta x\frac{\Delta t}{\Delta x}\left[6a + \frac{2}{\Delta x^2}(\cos k\Delta x - 1)\right]\right| \leq 1. \tag{76}$$

Now, note that $|\sin k\Delta x| \leq 1$ and that the largest magnitude that $|\cos k\Delta x - 1|$ can have is 2. Therefore, we have that

$$\frac{\Delta t}{\Delta x}\left[6a + \frac{4}{\Delta x^2}\right] \leq 1, \tag{77}$$

which is the stability condition we wanted to prove. The stability condition can easily be implemented in MATLAB by requiring that $\Delta t = \frac{(\Delta x)^3}{4}$.

The leading order terms for the truncation error can be obtained by using Taylor expansions. If we denote the numerical scheme by $\tilde{u}$ and the actual solution using u, the numerical scheme can be rewritten as

$$\begin{aligned}
\tilde{u}_t &= \frac{u(t+\Delta t)-u(t-\Delta t)}{2\Delta t} \approx u_t + \frac{1}{6}\Delta t^2 u_{ttt} \\
\tilde{u} &= \frac{u(x+\Delta x)+u(x)+u(x-\Delta x)}{3} \approx u + \frac{1}{3}\Delta x^2 u_{xx} \\
\tilde{u}_x &= \frac{u(x+\Delta x)-u(x-\Delta x)}{2\Delta x} \approx u_x + \frac{1}{6}\Delta x^2 u_{xxx} \\
\tilde{u}_{xxx} &= \frac{u(x+2\Delta x)-2u(x+\Delta x)+2u(x-\Delta x)-u(x-2\Delta x)}{2\Delta x^3} \approx u_{xxx} \\
&+ \frac{1}{4}\Delta x^2 u_{xxxxx}.
\end{aligned}$$

So, we are numerically solving

$$0 = \tilde{u}_t + 6\tilde{u}\tilde{u}_x + \tilde{u}_{xxx} \simeq u_t + 6uu_x + u_{xxx} + E(u), \tag{78}$$

where the scheme has truncation error $E(u) = \frac{1}{6}\Delta t^2 u_{ttt} + \Delta x^2\{2u_x u_{xx} + uu_{xxx} + \frac{1}{4}u_{xxxxx}\}$ [3].

When the numerical results from this method were compared to the exact results in Equation (9) in [3], it was found that all of the features of the soliton are accurate (compared to the exact solution), except for the soliton center, whose position can be corrected by replacing the center with

$$v = \frac{dx_c}{dt} = 4\eta^2 - \frac{4}{5}\eta^4\Delta x^2, \tag{79}$$

where the first term is the velocity of the unperturbed soliton, and the second term is the correction due to the truncation error [3]. Herman [3] analyzed this error as a perturbation of the KdV equation to show that the main effect of numerical error is the velocity given by

Equation (79).

Having selected a numerical scheme to simulate the KdV, we must decide how to simulate the noise term to allow comparison with the analytical results described by Wadati [9]. In the next chapter, we will look at the numerical realization Gaussian white noise and it's integral, Brownian motion.

5 NUMERICAL REALIZATION OF GAUSSIAN WHITE NOISE

We have already discussed the theory behind Gaussian white noise and Brownian motion. However, for computation purposes, we will need to use discretized Brownian motion, where $W(t)$ is specified at discrete t values. Let, $\delta t = T/N$ for some positive integer N and let W_j denote $W(t_j)$ with $t_j = j\delta t$. Previous conditions given for Brownian motion tell us that $W_0 = 0$ with probability 1 and

$$W_j = W_{j-1} + dW_j,\ j = 1, 2, \ldots, N \tag{80}$$

where each dW_j is an independent random variable of the form $\sqrt{\delta t}N(0,1)$. Here $N(0,1)$ denotes a normally distributed random variable with zero mean and unit variance.

We can easily implement discretized Brownian motion in MATLAB, using its generic random number function which produces an independent "pseudorandom" number from the $N(0,1)$ distribution. The numbers generated by the random number generator are scaled by $\sqrt{\delta t}$ and are used as increments in the FOR loop that creates the numerical array W, which contains the cumulative sums of dW.

We can graph the path for one realization of Brownian motion. A sample MATLAB code for a discretized Brownian path on the interval $t = 0 \ldots 1$ with $dt = .002$ is given by the following [5]:

```
randn('state',100)            % set the state of randn
T = 1; N = 500; dt = T/N;
dW = sqrt(dt)*randn(1,N);     % increments
W = cumsum(dW);               % cumulative sum
```

Figure 5 is an example of Brownian motion over the interval $[0, 1]$ with the number of points $N = 500$. This code uses MATLAB's random number generator (called *randn*) to produce independent "pseudorandom" numbers from the N(0,1) distribution. The state command

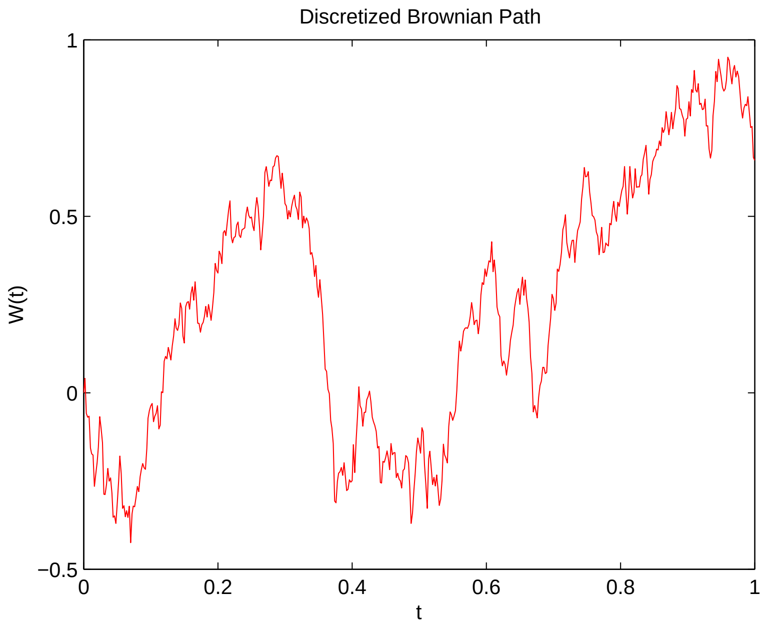


Figure 5: Discretized Brownian path over $[0, 1]$ with $N = 500$ time steps.

allows us to generate the same numbers each time. To generate different sets of numbers we remove the state command. The numbers from *randn* are scaled by $\sqrt{\delta t}$ and W is created as an array of the cumulated sums of each of these numbers. In Figure 5 we plot $W(t)$ vs. t and the discrete data points have been connected with lines so the discretized Brownian path can easily be visualized.

We recall that given a suitable function $h(t)$, the integral $\int_0^T h(t)dt$ may be approximated by the Riemann sum

$$\sum_{j=0}^{N-1} h(t_j)(t_{j+1} - t_j), \tag{81}$$

over the discrete points $t_j = j\delta t$. Similarly, we can consider a sum of the form

$$\sum_{j=0}^{N-1} h(t_j)(W(t_{j+1}) - W(t_j)), \tag{82}$$

which can be regarded as an approximation to the stochastic integral $\int_0^T h(t)dW(t)$. In this instance, $h(t)$ is being integrated with respect to Brownian motion.

The "derivative" of W with respect to t is

$$\frac{dW}{dt} = \eta. \tag{83}$$

We are interested in finding dW, so we see that

$$dW = \eta dt. \tag{84}$$

This value is called Gaussian white noise, and it is the quantity in which we are interested. It can be approximated by $\Delta W = W(t_{j+1}) - W(t_j)$, as seen in (82).

A stochastic process $\{X(t), t \geq 0\}$ is called *Gaussian*, or *normal*, if the random variables $W(t_1), \ldots, W(t_n)$ have a multivariate normal distribution for all $n \geq 1$, $t_1, \ldots, t_n$ [7]. Therefore, Brownian motion (and it's derivative) is also a Gaussian process, with each

$W(t_1), W(t_2), \ldots, W(t_n)$ being expressed as a linear combination of the independent normal random variables $W(t_1), W(t_2) - W(t_1), W(t_3) - W(t_2), \ldots, W(t_n) - W(t_{n-1})$. Also, $\{dW(t), 0 \leq t < \infty\}$ is white because it can be imagined that a time varying function $f(t)$ propagates through a white noise medium (where all frequencies are being "played") to yield the output $\int_b^a f(t)dW(t)$ [7]. This function will have zero autocorrelation over space.

Before we introduce the Gaussian white noise into the Zabusky-Kruskal scheme, we must first confirm that our MATLAB code upholds the properties of Brownian motion (i.e. standard Wiener process) and Wadati's identities that have previously been mentioned. In Figure 6, we can see that this process has empirical mean close to zero and a variance $t - s = t$ for $s = 0$, as expected. For times larger than 1000 we could expect the sample variance to be even closer to a straight line with slope of one.

Next, we want to confirm Wadati's identities in Equations (44) and (45) [9]. The following snippet of MATLAB code is used to confirm these identities (with varied parameters):

```
N=2000; eps=0.1; mu=sqrt(2*eps); s=0; dt=0.1; t1=5; t2=8;
k1=round(t1/dt+1); k2=round(t2/dt+1);
for j=1:N
    t=(j-1)*dt;
    r=randn(N,1);
    w=zeros(N,1); w(1)=w0;
    for i=2:N
        w(i)=w(i-1)+mu*sqrt(dt)*r(i);
end s=s+w(k1)*w(k2); end s=s/N
```

The first identity we would like to investigate is given by

$$< W(t_1)W(t_2) >= 2\epsilon \min(t_1, t_2).$$

For the case $2\epsilon = 1$, with times $t_1 = 5$ and $t_2 = 8$, we expect $< W(5)W(8) >= 5$ over many

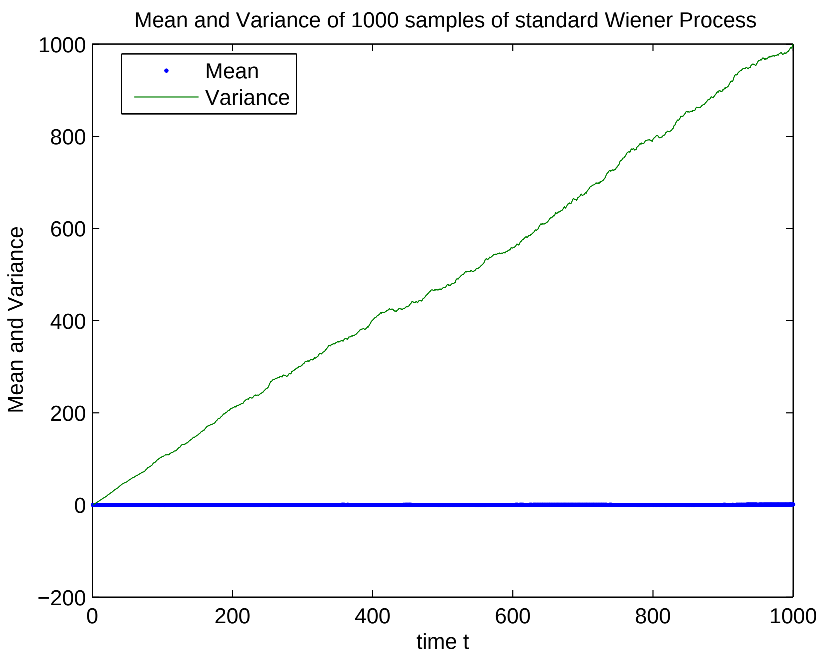


Figure 6: Mean and variance of 1000 samples of a standard Wiener process.

realizations of the noise. When we processed the previous code 1000 times we received an answer of 4.9122. Next, we want to introduce the multiplier of $\mu = \sqrt{2\epsilon}$ in order to give the right expected value for products of Wiener processes. Using the previous code with $\epsilon = 0.1$ should give us $< W(t_1)W(t_2) >= 2(0.1)5 = 1.0$. The answer we find is 1.0296. Because there is some random deviation, we would like to see if taking the mean over several more runs (2000) will help get us closer to 1. Using the previous code snippet, we arrive at an answer of 1.0296. Finally, to check the expected value for general times (with $dt = .1$), we use the previous code and found an answer of 1.0113 (which is even closer to 1).

The other identity to confirm is

$$< \exp(cW(t)) >= \exp(\frac{1}{2} < W^2(t) >).$$

We see the results of our MATLAB code in Figure 7. In Figure 7 we can see that this identity is approximately confirmed. If we increased the number of runs the lines should be almost indistinguishable.

In the next chapter we will show how we can present our simulation results for the KdV equation with and without noise.

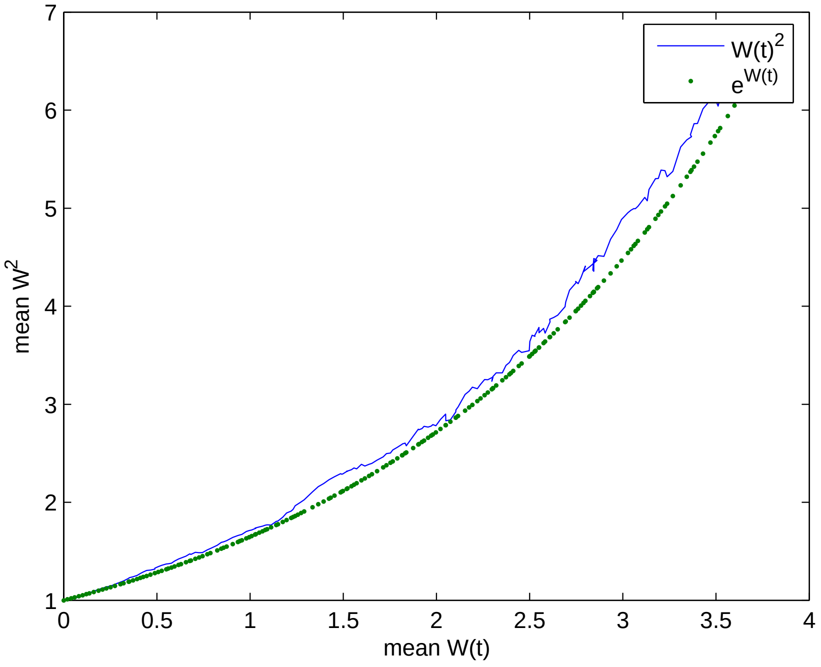


Figure 7: $< \exp(cW(t)) >$ vs. $\exp(\frac{1}{2} < W^2(t) >)$ confirming Wadati's identity.

6 STOCHASTIC KDV CODE

From the beginning, we have been interested in comparing our results to Wadati's ([9] and [10]) for large times ($t \to \infty$). Unfortunately, we found that our initial code for the Zabusky-Kruskal scheme (herein ZK scheme) required at least 90000 time steps (using varied parameters), in addition to the code for calculating the exact solution for comparison reasons (otherwise we would not be following the soliton long enough). On a computer with a 2.6 gHZ Intel P4 processor and 512 MB of RAM, our initial code required nearly 50 hours. To fully utilize the computing resources we have, it is therefore necessary to explore how MATLAB allocates memory and computes expressions using loops.

The following is a snippet of the initial code we wrote to compute each solution of the stochastic KdV:

```
for count=1:runs
    % Generate Gaussian noise
    for j=1:Tsteps
        dW(j)=mu*sqrt(dt)*randn;
    end;
    % Time Loop
    for i=3:Tsteps
      % Generate solution using Zabusky-Kruskal scheme
      for j=3:N-1
       u2(j)=u0(j)-((2*dt*u1(j+1))/(dx))*(u1(j+1)-u1(j-1)+...
       u1(j)+u1(j-1)-1/(dx^2))+((2*dt*u1(j-1))/(dx))*...
       (u1(j)+u1(j-1)-1/(dx^2))-(dt*u1(j+2))/dx^3+...
       (dt*u1(j-2))/dx^3+(dW(i)+dW(i-1));
    end;
end;
```

Of course this is only part of the code, but we are interested in looking at the repeated calculations required when we implement the ZK scheme in this way. In this snippet, we see that there are three nested for loops that must be run for MATLAB to compute one solution of the KdV equation using the ZK scheme. We also see that there is a fourth loop required to generate Gaussian noise. If the arrays (u0,u1,u2,dW) are not initialized in order from largest to smallest, not only could it take MATLAB up to 50 hours to complete 500 runs, but we would be limited by the computer's RAM.

Fortunately, MATLAB supports vectorization (as seen at the MATLAB homepage), or the process of writing code that utilizes matrix operations or other fast built-in functions instead of using time consuming loops. The following is a sample of the code we used for vectorization:

```
A=(spdiags(-2*ones(N1),1,N1,N1)+spdiags(ones(N1),2,N1,N1)
...+spdiags(2*ones(N1),-1,N1,N1)
...+spdiags(-ones(N1),-2,N1,N1))*dt/dx^3;
B=(spdiags(ones(N1),1,N1,N1)+spdiags(-ones(N1),-1,N1,N1))*dt/dx;
C=(spdiags(ones(N1),1,N1,N1)+spdiags(ones(N1),0,N1,N1)
...+spdiags(ones(N1),-1,N1,N1))*2;
D=spdiags(ones(Tsteps),0,Tsteps,Tsteps)+...
spdiags(ones(Tsteps),-1,Tsteps,Tsteps); for chunk=1:NChunks


  for i=istart:Tsteps
    for count=1:runs


    dW=mu*sqrt(dt)*randn(1,Tsteps);
    % Generate solution using Zabusky-Kruskal scheme with
    % vectorization.
    % The separate terms are given in vectorized form as
    %   UXXX=A*u1;
    %   UX=B*u1;
    %   U=C*u1;
    %   u2 = uu0-UXXX-U.*UX;
    u2 = u0 -A*u1-(C*u1).*(B*u1)+ddW(i)*ones(size(u0));
    end;
  end;
end;
```

Here u0 is the solution with the initial condition, u1 is the first solution after the initial condition, and u2 is used to compute the rest of the solution of the ZK scheme. UXXX is given by A*u1, which means that the first solution after the initial is multiplied by the matrix A (which sets up sparse matrices for the actual ZK scheme). The other terms of the KdV equation are computed similarly. We decided to use chunks and timesteps so that the individual matrix computations could be handled by MATLAB. Otherwise, MATLAB's allocated memory would fill up before computation is complete.

This code does essentially the same thing as the previous code. There are still three FOR loops present in the code, but now all of the calculations are done using the matrix operator .* (which performs cumulative multiplications on all elements of the array at one time instead of one at a time). All of the arrays are now initialized earlier in the code from biggest to smallest, which aids MATLAB's memory allocation since the smaller arrays can fit in the "holes" in memory created by the bigger ones. Also, pre-allocation of arrays prevents MATLAB from continuously setting up arrays during the run. We also use sparse matrices in our setup to save space, since sparse matrices remove all extra zeros while retaining the general form and structure of the code.

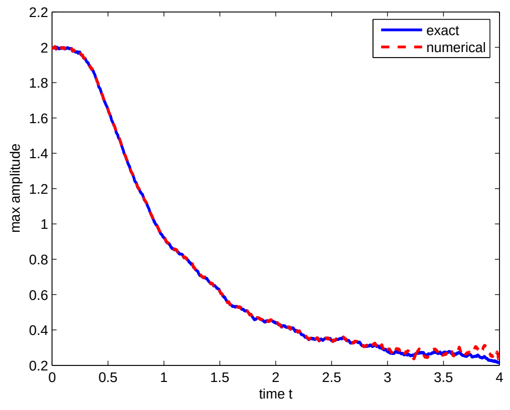
Using the code KDV TIME CHECK found in the Appendix, we find that running this code with 2000 time steps takes 1.859 seconds to complete with FOR loops and 1.125 seconds with vectorization (all parameters are the same for each block of code). While this does not appear significant, if we calculate how long it would take to do a run with 90000 time steps 500 times, we see that with FOR loops it will take 11 hours and with vectorization it will only take up to 5 hours. This is a big difference! Coupled with the fact that without proper array allocation running the code with FOR loops can take up to an estimated 50 hours for each parameter set explored. Vectorization is a very important thing to consider.

Now we must confirm that the ZK scheme gives us results close to those of the exact KdV ($u_t + 6uu_x + u_{xxx} = 0$) for the one soliton solution. To do this, we ran the vectorized code given in the appendix for one run. Figure 9 gives us the error between the numerical scheme and exact solution generated by the vectorized code. As we can see, when the code is considered on the interval $x \in [-10, 40]$ with 16000 time steps ($dt \approx .7$), the relative error between the ZK scheme and the exact solution is on the order of 10^{-3}. The small fluctuations seen in Figure 9 are present because the ZK scheme has a truncation error on the order of $O(\Delta t^2 + \Delta x^2)$. Therefore, the error is sufficiently small; we now have a scheme that can be used to simulate the KdV equation.

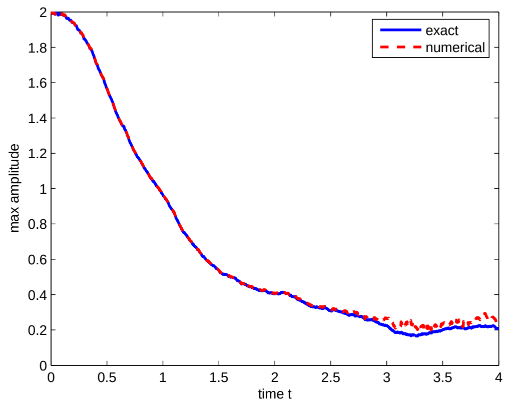
We are next interested in seeing how the ZK scheme compares to the exact solution when noise is introduced (as seen in equation (31)) and the ensemble average is computed. We processed the vectorized code for 500 runs, where a "run" is defined as one realization of the solution; these solutions are then averaged over the 500 runs. Each of these runs was done on the domain $[-10, 40] \times [0, 4]$ with 16000 time steps (using $dt = \frac{dx^3}{4}$), as was done before, and the L^∞ norm of the maximum value of the differences in amplitude for both the ZK scheme and the exact solution was found. The following table shows the error of the ZK scheme when compared to the exact solution for each given ϵ:

In Table 1 the norm column contains theL^∞ norms of the differences of the numerical and exact data. We can see that the errors begin on the order of 10^{-2} and are slightly larger or smaller as ϵ gets smaller and smaller (as expected, as smaller ϵ means less influence on the numerical scheme).

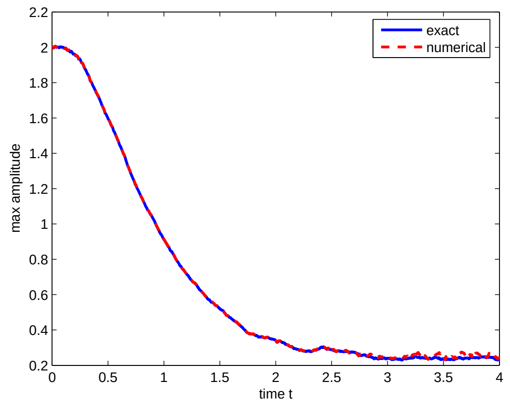
Having succeeded in improving the efficiency of the code, we must decide how many runs



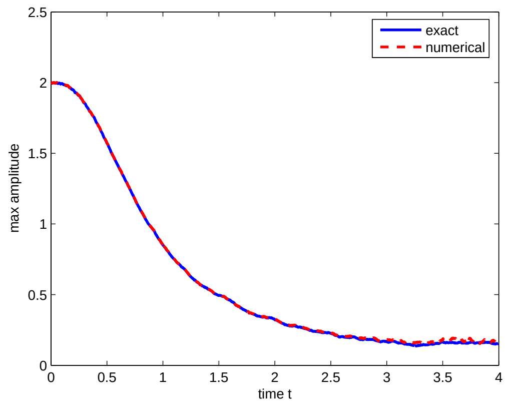
(a)



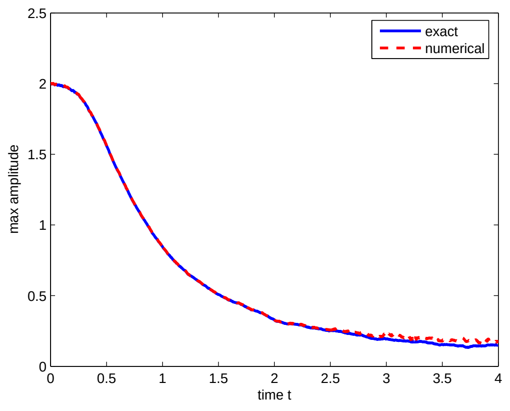
(b)



(c)



(d)



(e)

Figure 8: Amplitude decay using the ZK scheme and the exact solution for $\eta = 2$ and $\epsilon = 0.01$ showing (a) 100 runs through (f) 500 runs.

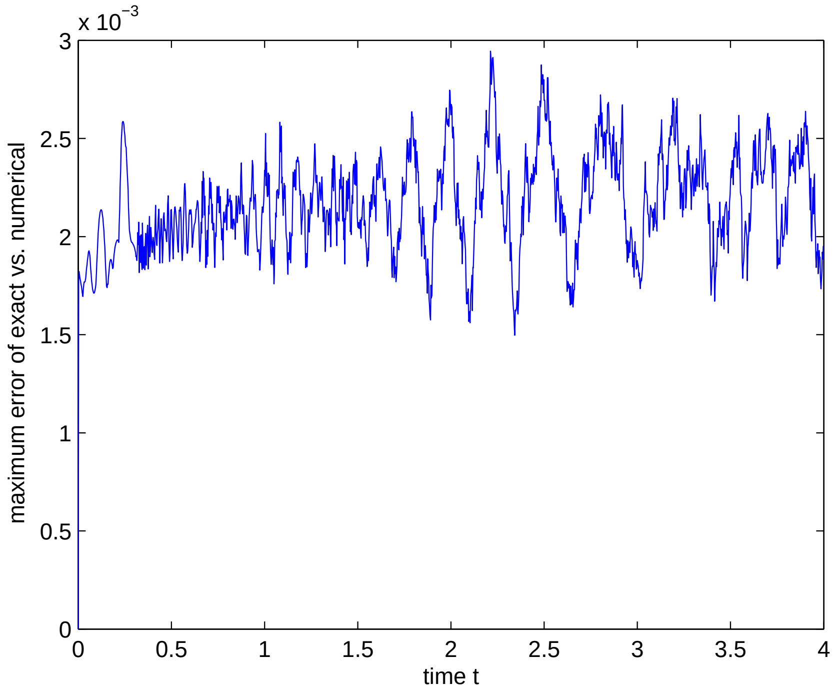


Figure 9: Maximum Error comparing the ZK scheme to the Exact solution with velocity correction.

Numerical Vs. Exact	
Epsilon	Norm
.1	0.023
.01	0.005
.001	0.009
.0001	0.006
.00001	0.006
.5	0.067
.05	0.010
.005	0.003
.0005	0.007
.00005	0.006
.25	0.018
.025	0.011
.0025	0.004
.00025	0.006
.000025	0.006

Table 1: Absolute Norms of errors of Numerical vs. Exact.

to do to get the smallest error possible. Figure 8 justifies why we chose to do 500 runs to get the best possible data. These runs were done on the domain $[-10, 40] \times [0, 4]$ for 16000 time steps with an ϵ of 0.1. As we progress from (a) to (e), we see that the ZK scheme's amplitude begins to converge closer and closer to the exact solution with minimum amount of error. The best fit is seen in figure (e), where 500 runs were completed. We decided to keep 500 runs as the maximum number of runs because of the error and computing time constraints.

7 RESULTS

We present here the results of our simulations using the vectorized code. As we can see from Figure 8, the amplitude of the soliton decreases largely by the end of the interval we are observing. Therefore, it seems that Wadati's "diffusion" does appear in the results. However, more experimentation is required to verify Wadati's prediction of a decay on the order of $t^{-3/2}$ for no damping and $t^{-1/2}\, e^{-\gamma t}$ for damping.

Figure 10 shows 500 runs for several values of ϵ and η, over the space interval $[-10, 40]$. As we can see in Figures 10 (a-b), with $\eta = 1.5$, there is almost no difference between the ZK scheme and the exact solution with varied ϵ. However, in Figures 10 (c-d), with $\eta = 2.0$, there is some deviation around $t = 2.5$. This deviation is not seen when we increase the x-interval to a new interval of $[-10, 90]$ and increase the number points sampled from $N = 500$ to $N = 1000$ (this way dx and dt is not affected), as depicted in Figure 11. Here, the numerical solution is closer to the exact solution. So, any time the soliton has $\eta = 2.0$ (or greater), a larger x-interval needs to be used to reduce the error in averaging. This is necessary because the soliton travels faster and some "walk" off due to Brownian motion.

Now we want to verify that the decay of these averaged solutions is on the order of $t^{-3/2}$ for no damping as Wadati predicted. Figure 12 shows a log-log plot of 500 runs with maximum amplitude versus the time of the averaged numerical solution (with $\epsilon = .05$ and $\eta = 1.5$ for 16000 time steps): As we see in Figure 12, the amplitude exhibits a power law behavior for $t > 1$ which satisfies $t^3 >> \frac{1}{48\eta^2\epsilon}$. We want to look at the linear portion of the log-log plot and use MATLAB's polynomial fit function so we can estimate the slope. Namely, if $y = At^b$, then $\ln y = b \ln t + \ln A$. Therefore, the slope of the linear fit for $\ln y$ vs, $\ln t$ gives us the decay constant b that were are seeking.

We found the linear regression line using MATLAB's polynomial fit function. We see in Figure 13 that the lines are almost indistinguishable, and from the linear fit we get a slope of $-1.47 \pm .01$ (this was found using the formula for uncertainty found in [4]). The confidence

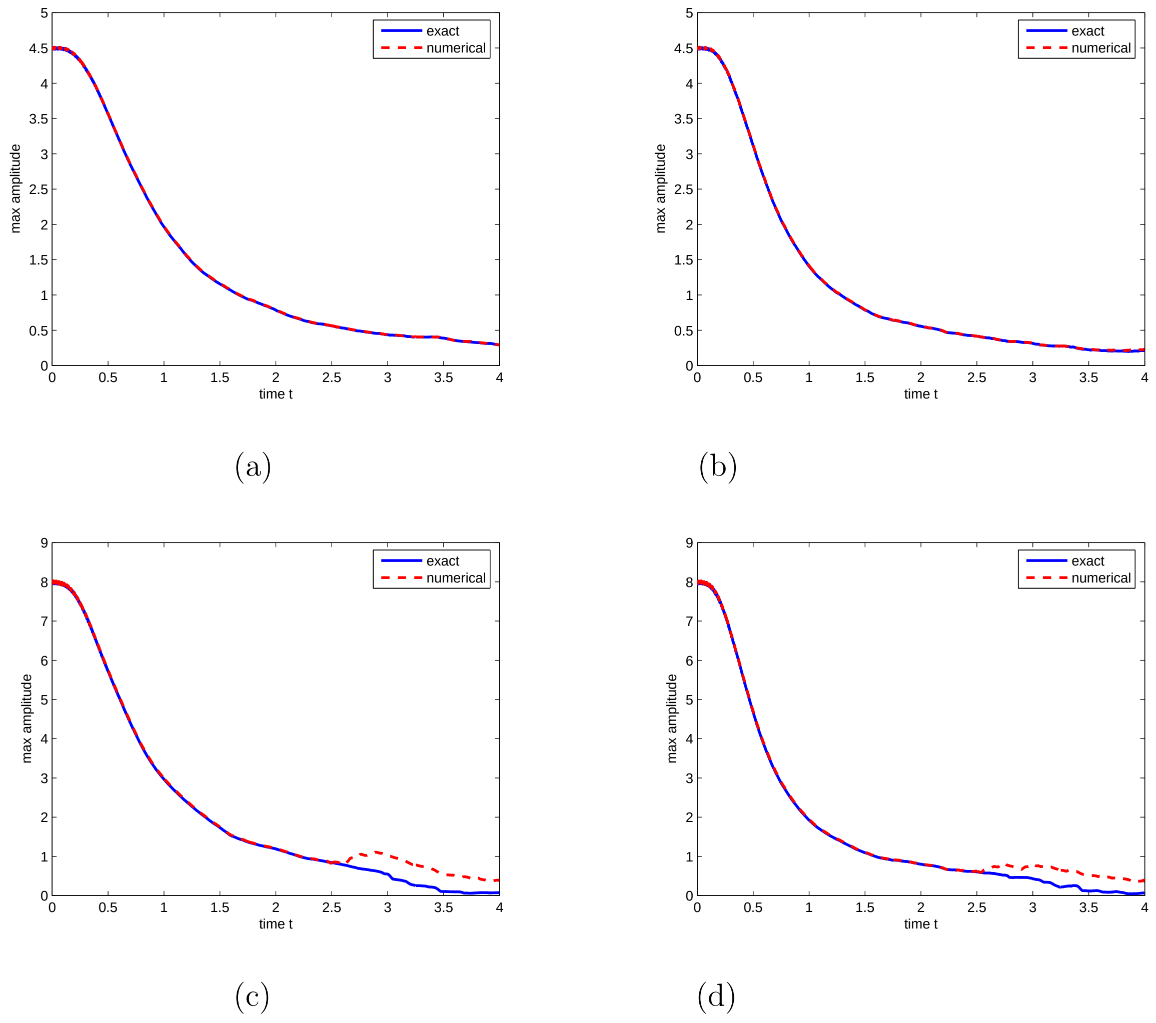


Figure 10: Plot of $< u(x,t) >_{max}$ vs. t for both the exact and numerical solutions to the KdV for parameters: (a): $\eta = 1.5$ and $\epsilon = 0.05$, (b): $\eta = 1.5$ and $\epsilon = 0.1$, (c): $\eta = 2$ and $\epsilon = 0.05$, (d): $\eta = 2$ and $\epsilon = 0.1$.

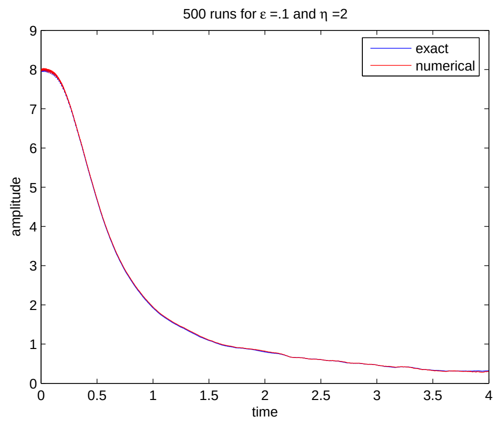


Figure 11: Amplitude vs. time plot for 500 runs on a larger x-interval ($[-10, 90]$) with $\epsilon = .1$ and $\eta = 2$.

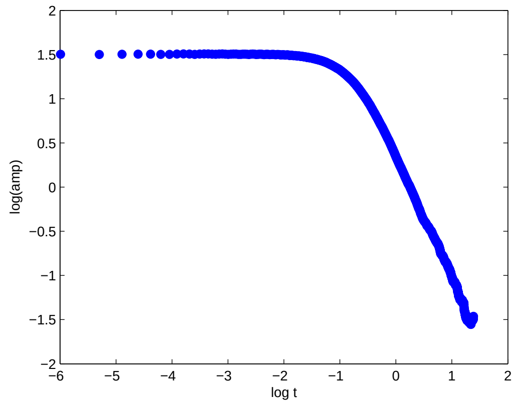


Figure 12: Plot of log(amplitude) vs. log(time) for data in Figure 11.

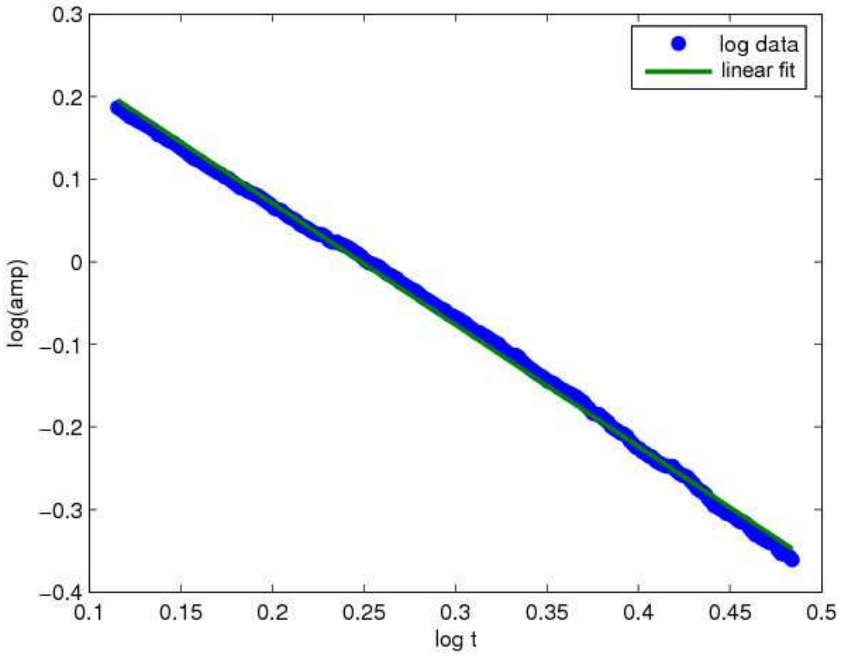


Figure 13: Linear portion of the log-log plot of amplitude vs. time with a linear fit for data in Figure 11.

interval is based on an r^2 value of .9879, which means that our linear fit explains 98% of the variation of the log-log values. Therefore, Wadati's prediction of the decay order being $t^{-3/2}$ is seen in our numerical simulations.

We next looked at how damping effects the amplitude of the soliton with and withouse noise. Recall the theoretical solution in Equation (64)

$$< u(x,t) >= \frac{\eta^2}{\sqrt{\pi b}} \int_{-\infty}^{\infty} e^{-(a-s)^2/4b} \operatorname{sech}^2 \frac{s}{2} \, ds, \tag{85}$$

where

$$a = 2\eta(x - x_0 - 2\eta^2 t) \tag{86}$$

and

$$b = \frac{36\eta^2}{\epsilon\gamma^2} \left[2\epsilon\gamma t - 3 + 4e^{-\epsilon\gamma t} - e^{-2\epsilon\gamma t} \right] \tag{87}$$

First, we want to look at the integral solution for $< u(x,t) >$ in Equation (85) to see how noise and damping both effect the amplitude decay of the soliton. We can find the peak $< u(x,t) >_{max}$, by setting $a = 0$ and numerically integrating. This is done by using MATLAB's quad function and we found that our numerical integral produced little error.

In Figure 14 we show some of our results for different values of γ and ϵ. We start with small γ and small ϵ in the upper left hand corner and progress to large γ and large ϵ in the bottom right hand corner. As is expected, when small damping and small noise are present, the amplitude decays slower than in the presence of larger damping and larger noise. The interesting fact in Figure 14 is that in the presence of larger damping and little noise (for example $\gamma = 0.3, 0.5$ and $\epsilon = 0.01, 0.1$), it appears that the amplitude's decay does not depend on the noise as much as it does on the damping. We will verify that the numerical solution of the stochastic KdV also exhibits this behavior.

As we have done before, we now want to explore the results we get when we apply our modified Zabusky-Kruskal numerical method to the damped, stochastic KdV equation and

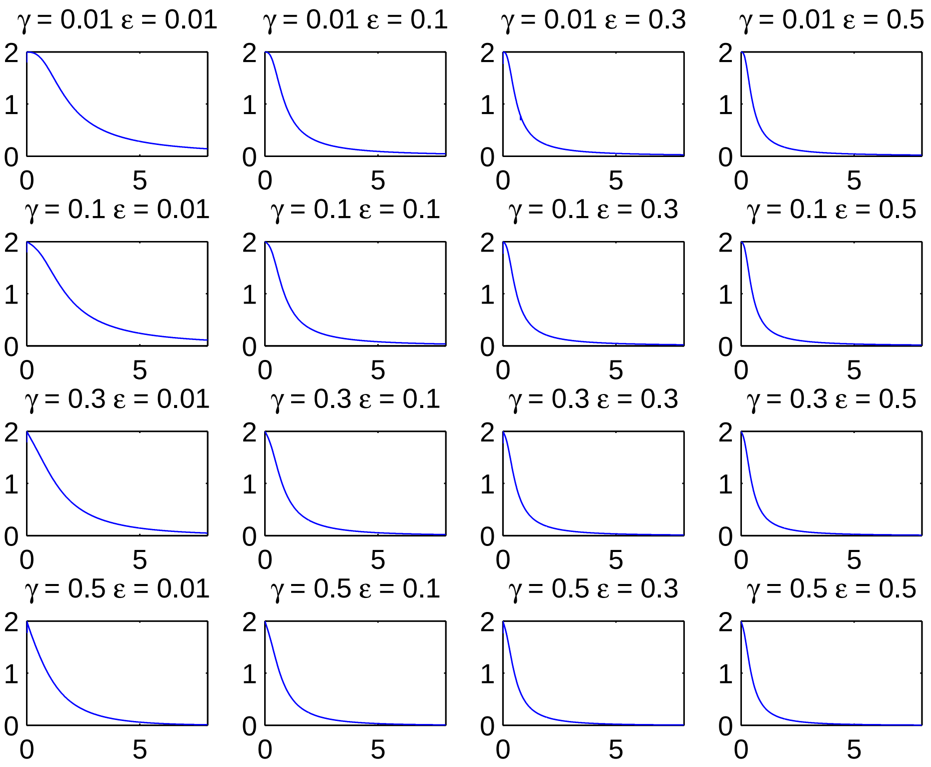


Figure 14: Amplitude vs. time plots for different values of γ (damping) and ϵ (noise) using the integral solution.

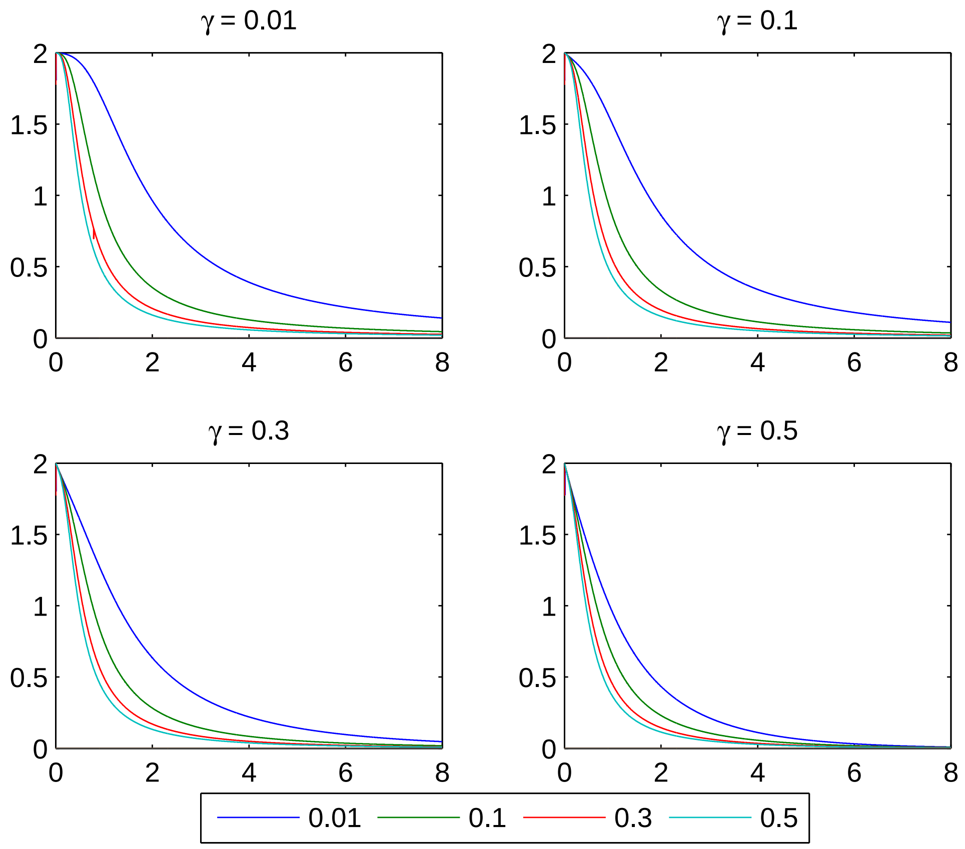


Figure 15: Amplitude vs. time plots for different values of γ (damping) and ϵ (noise; on the legend) using the ZK scheme.

take an ensemble average. We will look at different values of γ and ϵ to see if our numerical results are close to those of the integral solution in Equation (64). In Figure 15, we see that using small γ values and small ϵ (0.01) values leads to the slower decay of the soliton than for larger values of γ and ϵ (larger values are 0.5). As we increase the values of ϵ and γ, we see the decay at a faster rate. These results are exactly as we had expected based upon the integral solution in Equation (85) because larger γ means more decay.

Now we want to make sure noise effects the soliton in the way it should. In Figure 16, we see a group plot with varied ϵ values. The decay of the amplitude is the same as before, with larger values meaning faster decay. Figure 16 also shows us the large effect that noise has on damping, where a large presence of noise means faster decay.

Now that we are satisfied that the modified Zabusky-Kruskal numerical scheme yields results that are close to Wadati's integral solution for $< u >_{max}$, we want to make sure that the number of runs we chose (500) is sufficient for giving us good results. In Figure 17 we see that not only do the amplitude plots look like they should, but also that the curve obtained using 500 runs is indistinguishable from that using 1500 runs. Technically, using 1500 runs gives a sample average with smaller variance, but we also have to take computing constraints into consideration. Although it is harder to see the curves for large noise and damping levels, we can see that for small values of noise strength ($\epsilon = 0.01$) the solutions are visually close to each other.

We are finally ready to see if the decay order is close to the predicted value of $t^{-1/2}\, e^{-\gamma t}$ for damping. Figure 18 has a time interval of $[0, 10]$ and as we can see, small values of ϵ and small values of γ give us a decay order greater than $t^{-1/2}$, which is not what Wadati and Akustu predicted in [10]. However, for large values of ϵ and γ, we see that the decay order is a lot closer to $t^{-1/2}$. Therefore, we are interested in seeing how far out in time we should go to get a closer decay order of $t^{-1/2}$. In Figure 19 we display the integral solution for the time interval $[0, 100]$. The best result (-0.516) we obtained is for $\gamma = 0.5$ and $\epsilon = 0.5$. Even this value has a 3.1% error. Thus, we are not able to confirm the exact value for the decay

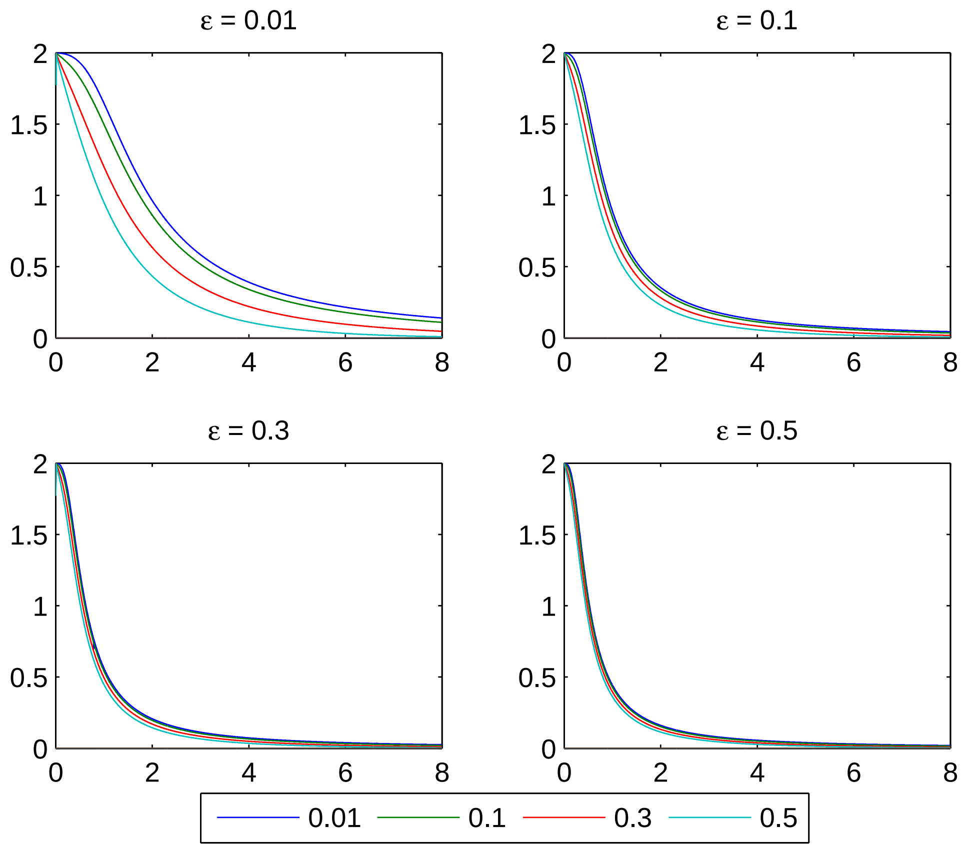


Figure 16: Amplitude vs. time plots for different values of ϵ (noise; on the legend) and γ (damping) using the ZK scheme.

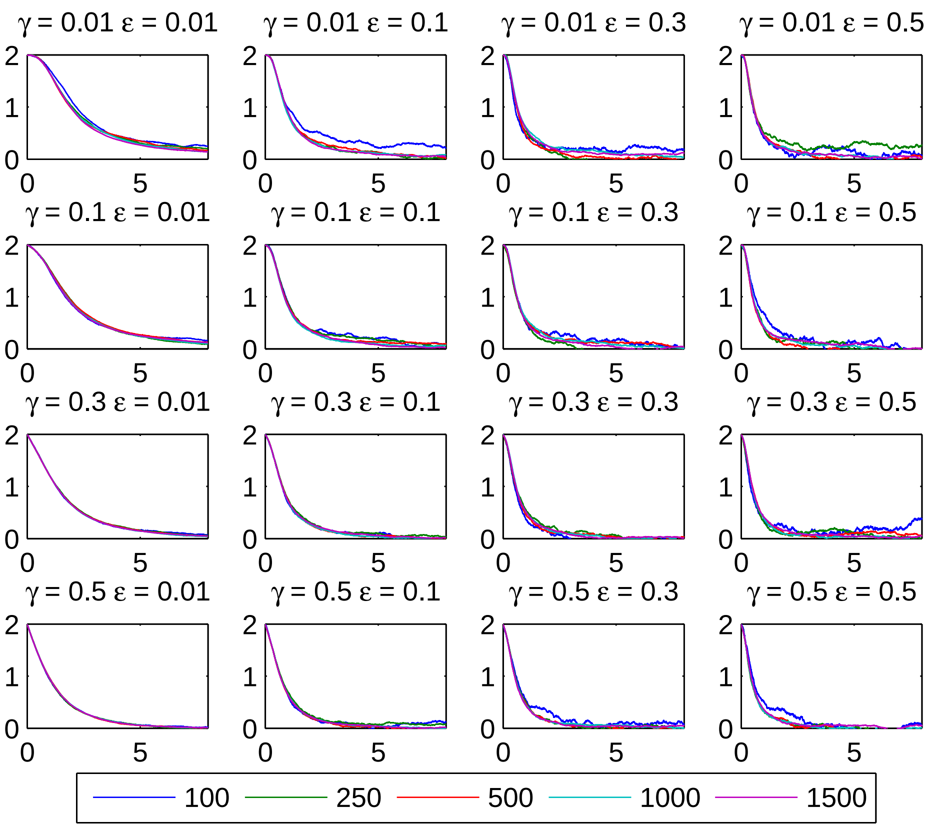


Figure 17: Amplitude vs. time plots for different values of ϵ and γ with number of runs on the legend.

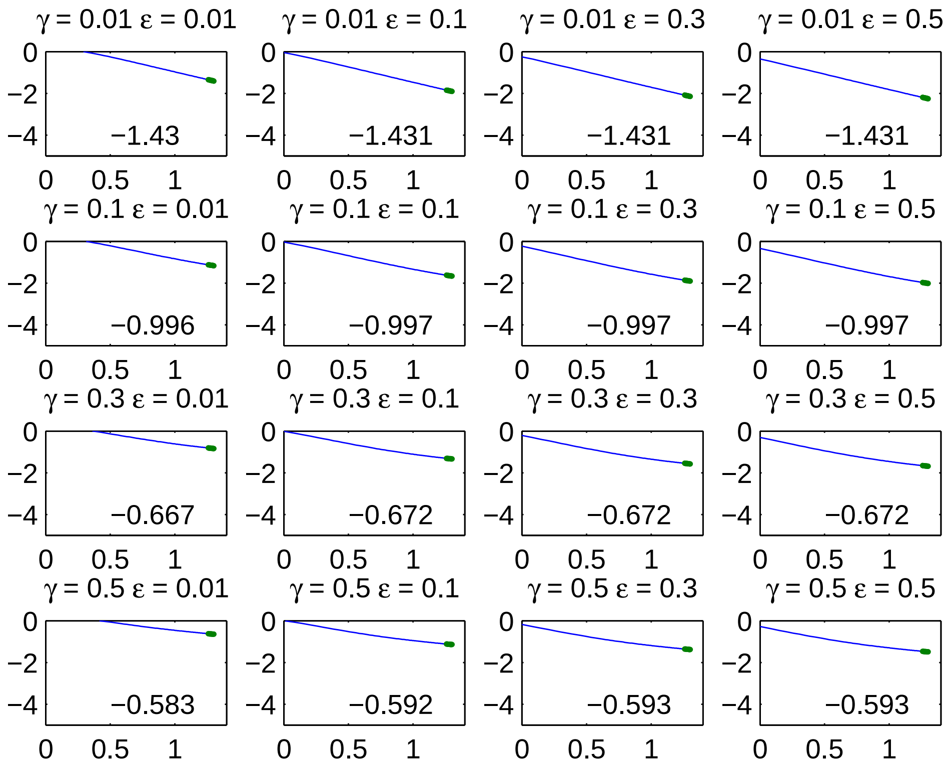


Figure 18: Power law decay rates for various realizations of noise and damping.

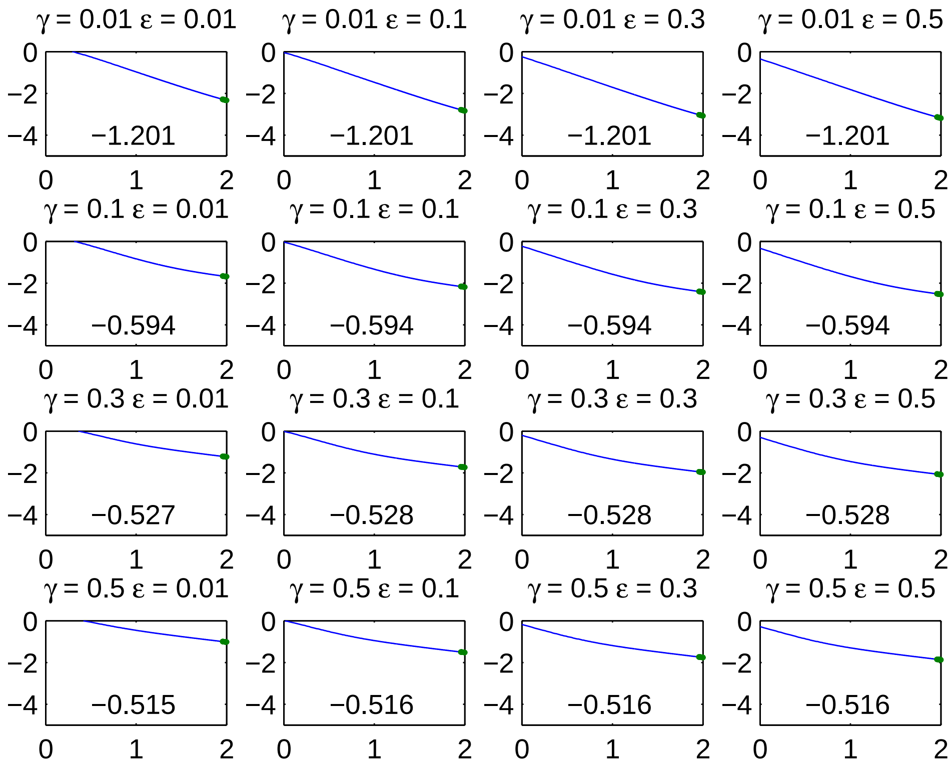


Figure 19: Power law decay rates for various realizations of noise and damping.

order, because we have to go so far out in time to get values close to $-1/2$; it is possible that the soliton has dispersed completely by then and we are only picking up background noise.

We still want to look at the numerical solution and see what we get for the decay order, so we do the same analysis that was previously done without damping. In Figure 20 we show a log-log plot for the data we have generated with 2000 runs for damping. We see that towards the end of the time interval that there is a lot of deviation. This is because we have not gone far enough out into time ($[0, 4]$) with our runs. So, we try and find a region where the graph is linear. In Figure 21 we notice that the regression line looks like it fits our data fairly well. We get a decay order of $-1.63 \pm .03$ (using the formula for uncertainty found in [4]). The r^2 value we are given for this linear fit is .9099, so 91% of the variation of our log-log data is explained by this linear fit. This is to be expected as our log-log data is not as smooth as it was before with our noise data.

We are also interested in trying our scheme for the two-soliton solution of the KdV equation. In Figure 22, we see that the behavior of the waves is exactly what we expect in the classical sense, the only thing that changes about the waves once they collide is their position in time, everything else about the wave stays the same. In Figures 23 and 24 we see that the Zabusky-Kruskal generated soliton solution follows the exact solution fairly closely, but some oscillations are present in Figure 23 due to numerical error. The general behavior of the ZK soliton remains the same however.

Now, we are interested in looking at the effect of damping and noise on the two soliton solution. As we can see in Figure 25, the two soliton solution with damping and noise follows the same basic path as the unperturbed two soliton solution. However, in Figures (26)-(28), we see that as ϵ and γ increase, the waves decay much quicker, sometimes before their collision is complete. Our numerical scheme has a problem with tracking the soliton since the soliton is basically nonexistent after its diffusion. In Figure 29, we plot the amplitude vs. time for the ZK soliton against that of the exact solution with the decaying amplitudes coming from the ZK scheme. We see that the amplitudes of both solitons decay in the same

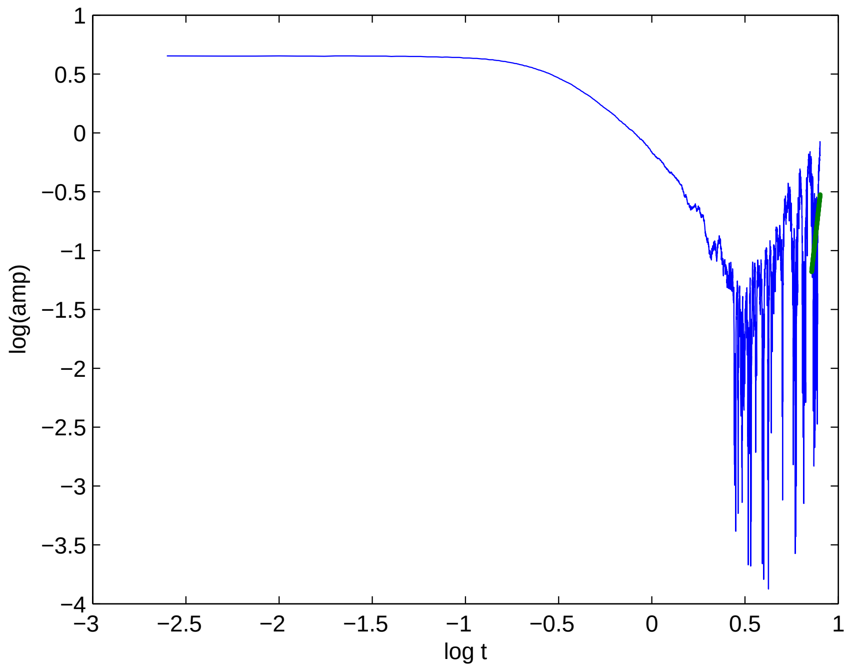


Figure 20: Plot of log(amplitude) vs. log(time) for $\gamma = 0.5$ and $\epsilon = 0.5$.

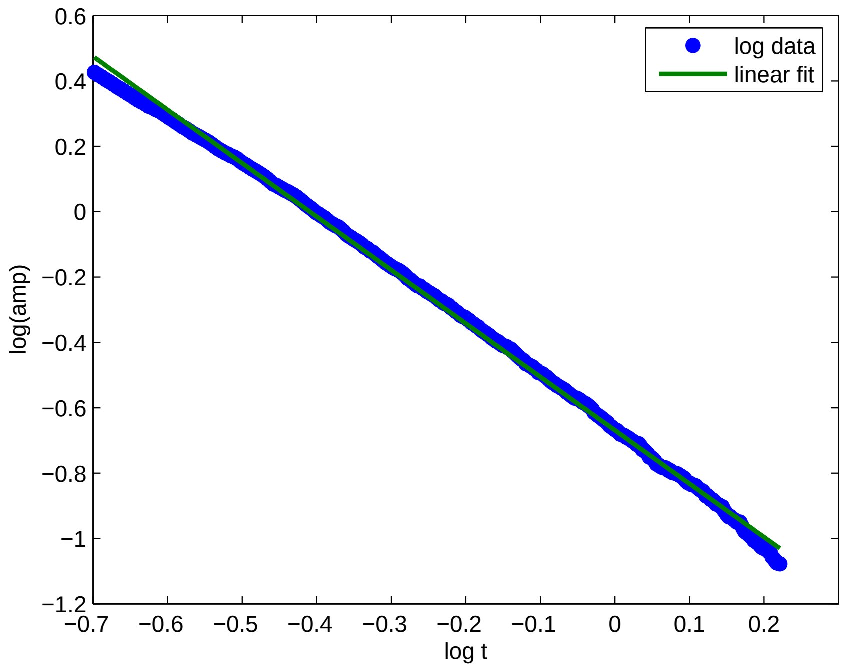


Figure 21: Linear portion of log-log plot of amplitude vs. time with linear fit for $\gamma = 0.5$ and $\epsilon = 0.5$.

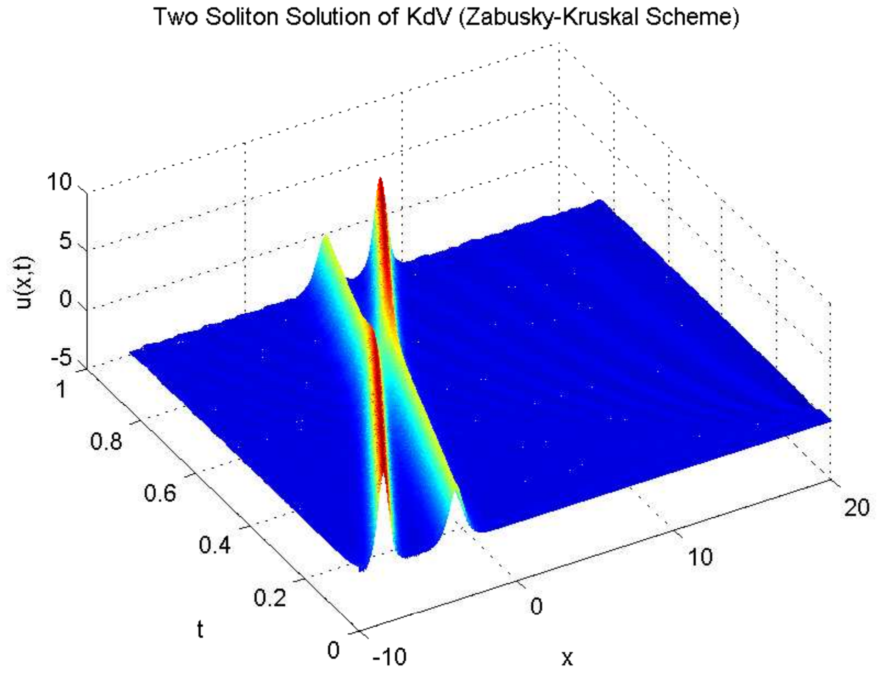


Figure 22: 3D plot of the two soliton solution using the Zabusky-Kruskal Scheme.

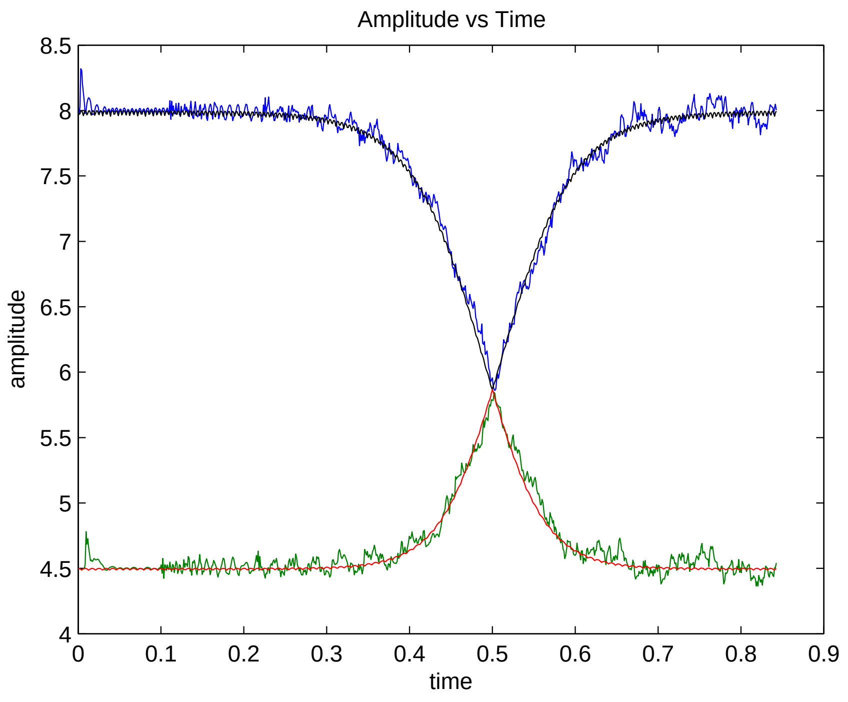


Figure 23: Amplitude vs. time for the two soliton solution using the Zabusky-Kruskal Scheme. Black and red represent the exact solution and blue and green represent the numerical solution.

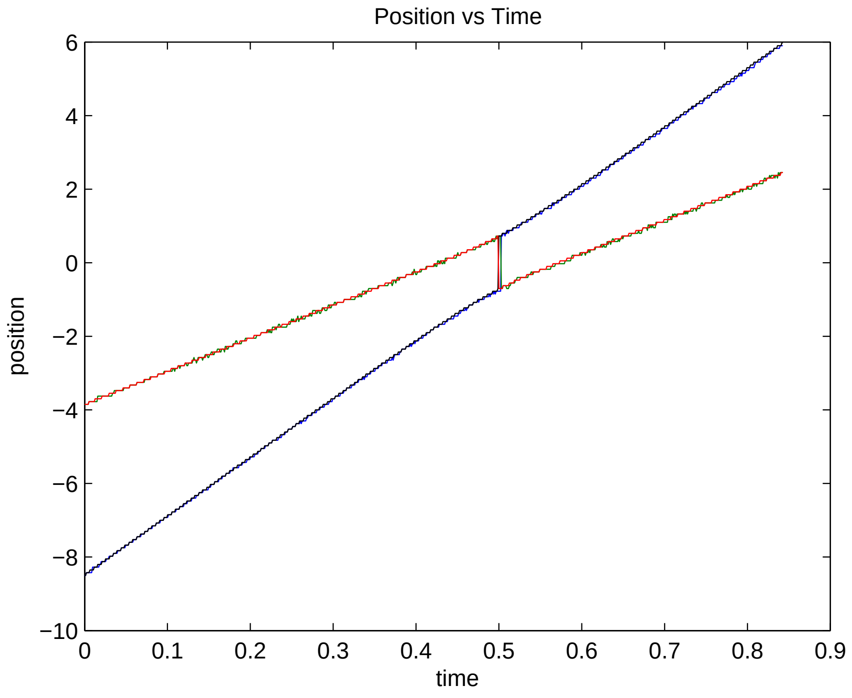


Figure 24: Position vs. time for the two soliton solution using the Zabusky-Kruskal Scheme showing soliton interaction. Black and red represent the exact solution and blue and green represent the numerical solution.

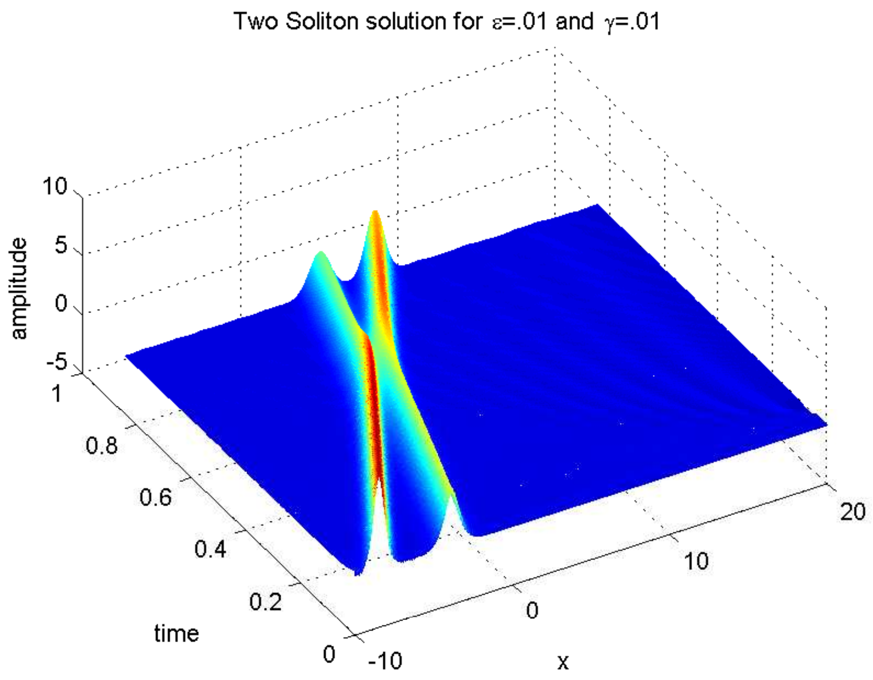


Figure 25: Damped stochastic two soliton solution for $\epsilon = 0.01$ and $\gamma = 0.01$.

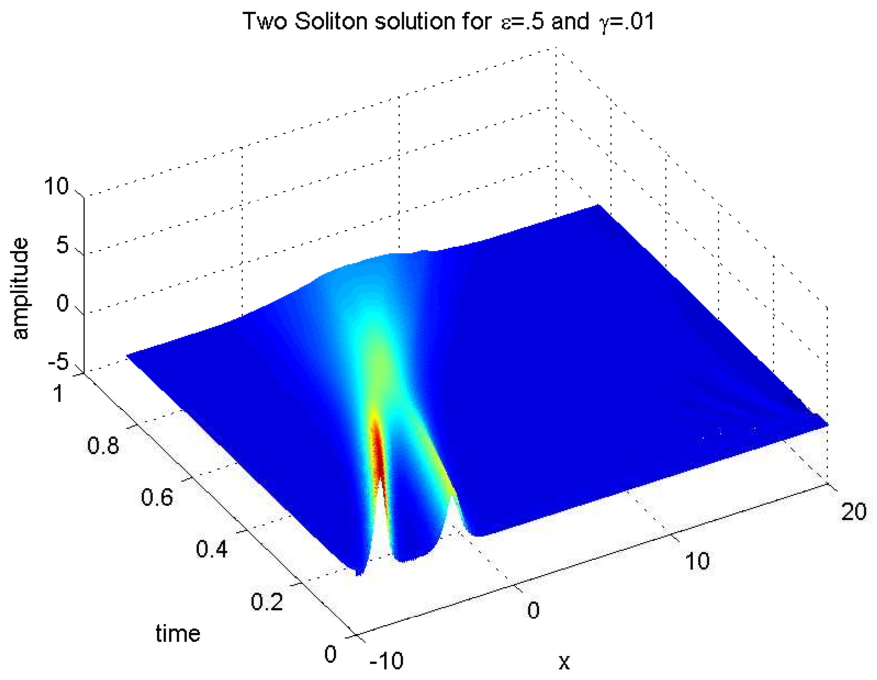


Figure 26: Damped stochastic two soliton solution for $\epsilon = 0.5$ and $\gamma = 0.01$.

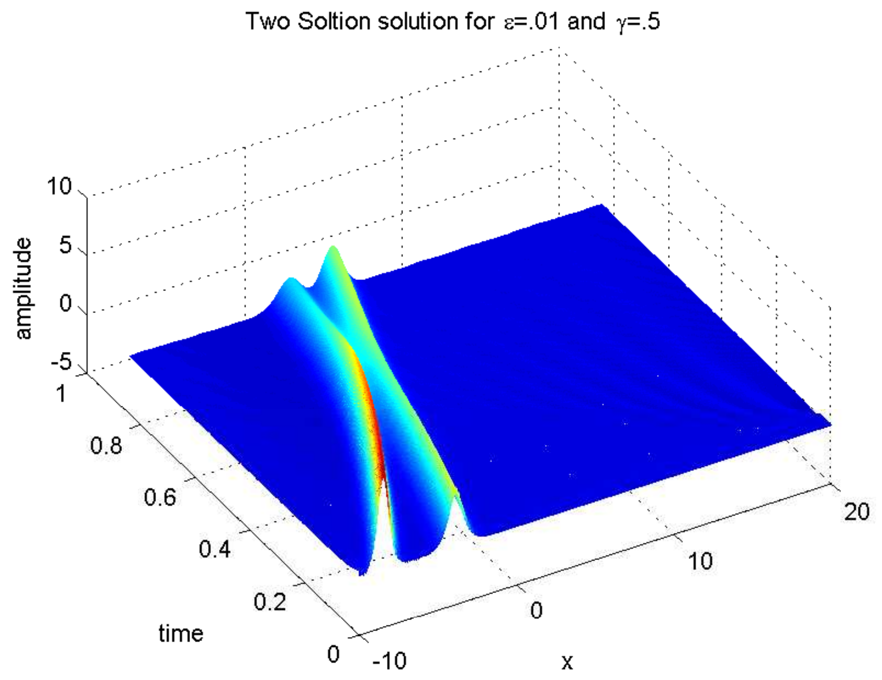


Figure 27: Damped stochastic two soliton solution for $\epsilon = 0.01$ and $\gamma = 0.5$.

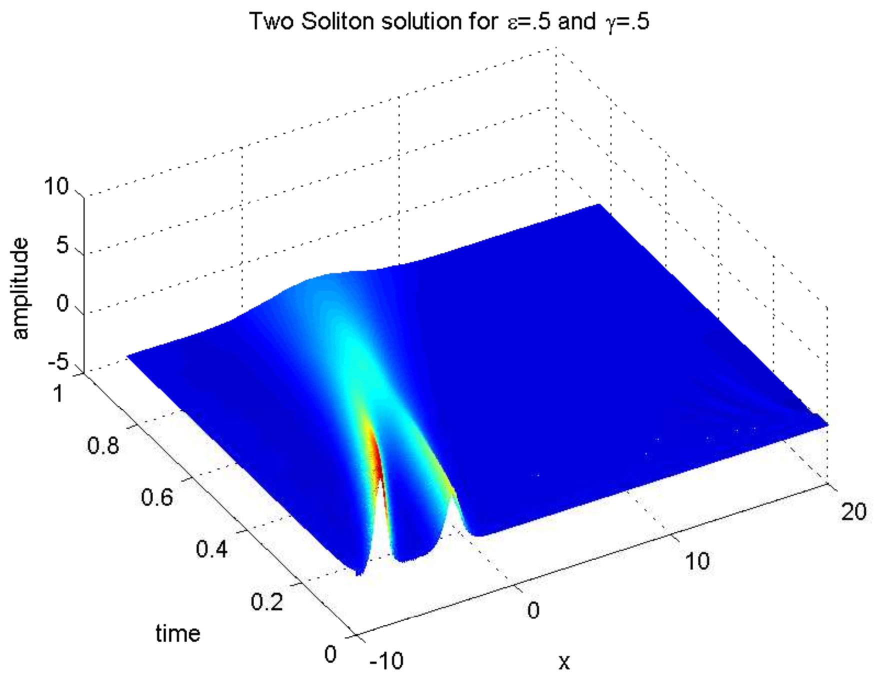
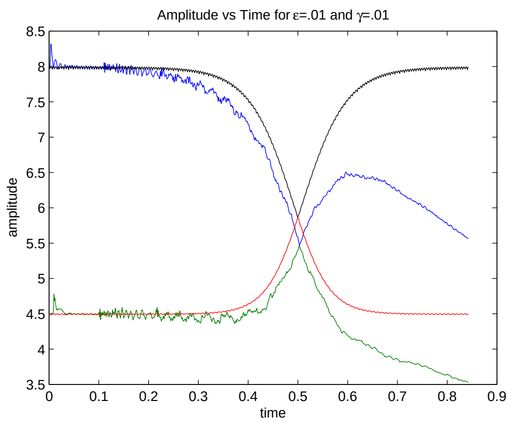
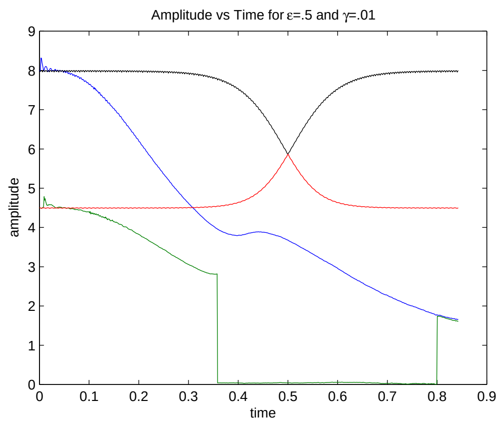


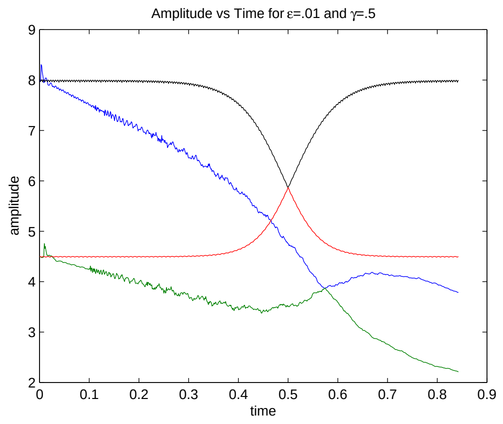
Figure 28: Damped stochastic two soliton solution for $\epsilon = 0.5$ and $\gamma = 0.5$.



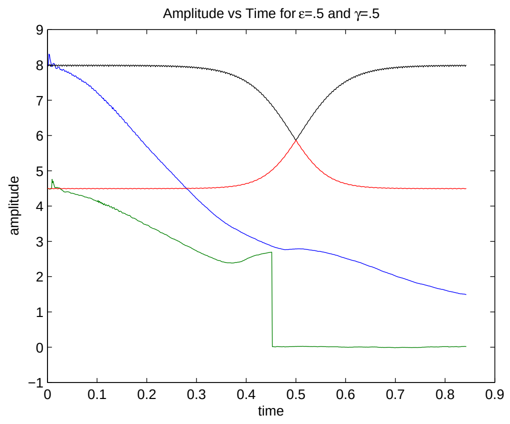
(a)



(b)



(c)



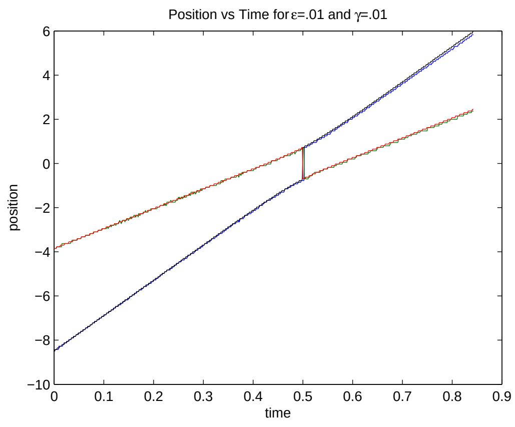
(d)

Figure 29: Amplitude vs. time showing two soliton interaction with varying values of ϵ and γ. Here black and red represent the exact solution and blue and green represent the numerical solution.

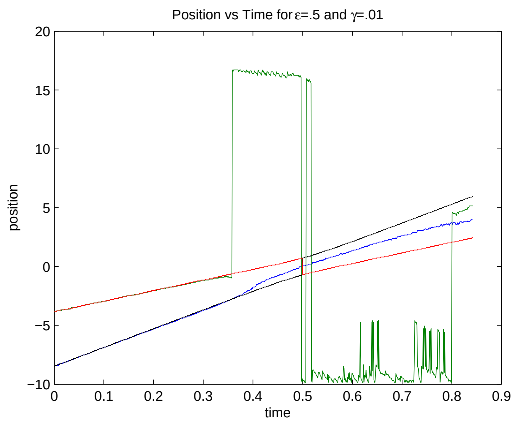
way as we saw for the one soliton solution: faster decay for large ϵ and γ. For large values of ϵ, the amplitudes of the wave appear to decay almost to zero. The severe oscillations in Figure 29 ((b) and (d)) are most likely due to the fact that we expected two peaks to be present; however, when the two solitons interact there is only peak present, so we pick up extraneous information. This is readily seen on the time interval $[0.3, 0.5]$ (near where the two solitons collide).

Finally, in Figure 30, we see that for small values of ϵ and γ, the positions of the solitons are almost exactly the same as their nonperturbed counterparts. However, for large values of ϵ and γ, the positions are displaced considerably. The explanation is that as the amplitudes of the solitons decay; the solitons will travel much slower or they are lost in the background noise. It is also much harder to follow both solitons in the region of the interaction or the peak range. This behavior is seen again with severe oscillations present on the time interval $[0.3, 0.5]$.

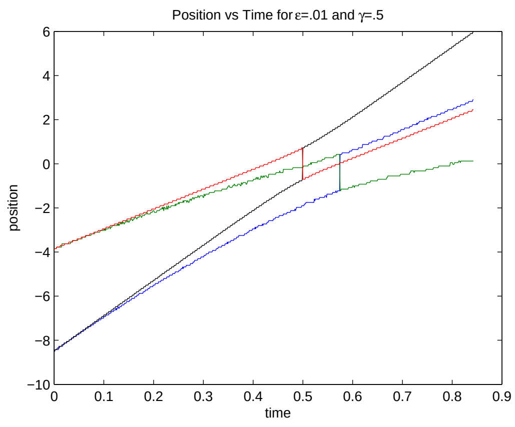
In conclusion, we found just like the one soliton soluton, the two soliton solution is affected by noise and damping in the same way, with more damping and more noise causing the waves to diffuse and disperse. We should only track the solitons when the peaks can be found so we can get an idea as to the area in which they interact.



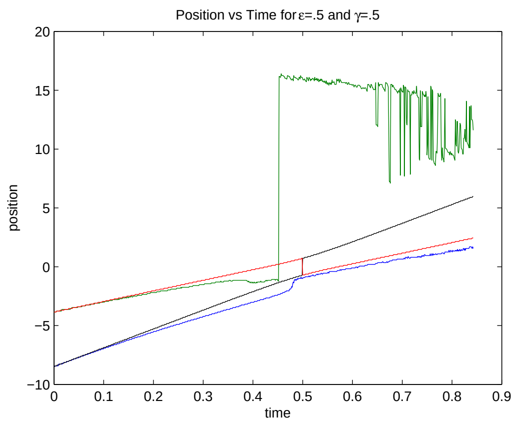
(a)



(b)



(c)



(d)

Figure 30: Position vs. Time showing two soliton interaction with varying values of ϵ and γ. Here black and red represent the exact solution and blue and green represent the numerical solution.

8 CONCLUSION

In this thesis, we have studied the KdV equation (1), the stochastic KdV equation with noise (2), the stochastic KdV equation with damping (3), and the two-soliton solution to the KdV equation (4).

We have found that the finite difference scheme for the KdV equation, the Zabusky-Kruskal method, as explained in Chapter 4, is accurate enough to be employed in simulating both the unperturbed KdV equation and the damped stochastic KdV equation, as long as there is a correction term added to take into consideration that the numerical soliton will be moving slower than the actual soliton [3].

We have also found that when employing the Zabusky-Kruskal scheme for the stochastic KdV, Gaussian white noise can be generated with little numerical error, and our results follow those found in Wadati [9]. Our numerical scheme produces results that follow his exact integral very closely, and that the decay found when looking at the average solutions of the noise perturbed solitons approaches $t^{-3/2}$, exactly as Wadati predicted. The Zabusky-Kruskal scheme also performs well when compared to the exact integral Wadati and Akutsu predicted in [10], and although we did not get exactly what they predicted for the decay order of the damped soliton ($t^{-1/2}\, e^{-\gamma t}$) we succeeded in showing that the soliton behaves exactly as predicted, both for the damping case and the damping with noise case. Vectorization of the code used to generate these numerical results is required as MATLAB is not used effectively otherwise.

Finally, we have found that for the exact two-soliton solution, the Zabusky-Kruskal scheme's results are close to those predicted for both noise and damping, as the waves follow the behavior outlined in [12] and [6].

Further avenues of research include looking at other numerical methods to simulate the stochastic KdV, further study of the damped soliton case to see why the slope of decay does not exactly match up for the numerical and exact, and studying different colors of noise to see their effect on the soliton. Another way to track the solitons in the two soliton solution should also be found so we can keep up with the peak even as it is lost in the background noise. It may be possible for our code to be expanded to use other wave forms as well.

REFERENCES

[1] P. G. Drazin and R. S. Johnson, “Solitons: an Introduction,” New York: Cambridge University Press, 1989.

[2] R. L. Herman, “Solitary Waves,” *American Scientist*, vol. 80, July-August 1992, 350-361.

[3] R. L. Herman and C. J. Knickerbocker, “Numerically Induced Phase Shift in the KdV Soliton,” *Journal of Computational Physics*, vol. 104, no. 1, January 1993, 50-55.

[4] Jack Higbie, “Uncertainty In The Linear Regression Slope,” *American Journal of Physics*, vol. 59, no. 2, 1990, 184-185.

[5] Desmond J. Higham, “An Algorithmic Introduction to Numerical Simulation of Stochastic Differential Equations,” *SIAM Review*, vol. 43, no. 3, 2001, 525-546.

[6] G. L. Lamb, Jr., “Elements of Soliton Theory,” New York: John Wiley & Sons, 1980.

[7] Sheldon M. Ross, “Introduction to Probability Models,” Sixth Ed., San Diego: Academic Press, 1997.

[8] A. C. Vliegenthart, “On Finite-Difference Methods for the Korteweg-de Vries Equation,” *Journal of Engineering Mathematics*, vol. 5, no. 2, April 1971, 137-155.

[9] Miki Wadati, “Stochastic Korteweg-de Vries Equation,” *Journal of the Physical Society of Japan*, vol. 52, no. 8, August 1983, 2642-2648.

[10] Miki Wadati and Yasuhiro Akutsu, “Stochastic Korteweg-de Vries Equation with and without Damping,” *Journal of the Physical Society of Japan*, vol. 53, no. 10, October 1984, 3342-3350.

[11] G. B. Whitham, “Linear and Nonlinear Waves,” New York: John Wiley & Sons, 1974.

[12] Tohru Yoneyama, “The Korteweg-de Vries Two-Soliton Solution as Interacting Two Single Solitons,” *Prog. Theor. Phys.*, V 71, # 4, April 1984, 843-846.

APPENDIX

ZK SCHEME CODE BEFORE VECTORIZATION

```
% This code uses the Zabusky-Kruskal scheme and Brownian motion to

% simulate the Stochastic KdV equation with white noise.


clear


% sKdV Parameters

eta=1.0; x0=5.0; a=-10; b=20; epsilon = 0.01;


% Run Parameters

runs=300; Tsteps=90000; N=500;


% Time and space increments

x=linspace(a,b,N+1); dx=(b-a)/N; dt=dx^3/4; t=(0:Tsteps-1)*dt;


% Noise initialization

mu=sqrt(2*epsilon); dW=zeros(1,N);


% Initialize Ensemble Average of Soliton Solutions

uu=zeros(Tsteps,N+1);


% Loop through several ensembles

for count=1:runs

    count
```

```
% Generate Gaussian noise
for j=1:Tsteps
    dW(j)=mu*sqrt(dt)*randn;
end;

% Brownian motion and its first integral
Iw=zeros(Tsteps,1);
w(1)=0;
Iw(1)=w(1)/2*dt;
for j=2:Tsteps
    w(j)=w(j-1)+dW(j);
    Iw(j)=Iw(j-1)+(w(j-1)+w(j))/2*dt;
end

% Calculate u0 and u1 from exact solution for first data points
u0=funckdv(x,0,eta,x0);
u1=funckdv(x,dt,eta,x0);
u2=zeros(size(u0));
uu(1,:)=uu(1,:)+u0;
uu(2,:)=uu(2,:)+u1;

% Time Loop
for i=3:Tsteps

   % Generate solution using Zabusky-Kruskal scheme
   for j=3:N-1
      u2(j)=u0(j)-((2*dt*u1(j+1))/(dx))*...
```

```
            (u1(j+1)-u1(j-1)+u1(j)+...
            u1(j-1)-1/(dx^2))+((2*dt*u1(j-1))/(dx))*...
            (u1(j)+u1(j-1)-...
            1/(dx^2))-(dt*u1(j+2))/dx^3+...
            (dt*u1(j-2))/dx^3+(dW(i)+...
            dW(i-1));
         end;

         % Set boundary conditions to account for
         % background noise level
         % and forcing first derivatives to
         % vanish on the ends

         u2(1)=mean(u2(N-1-N/50:N-1));
         u2(2)=u2(1);
         u2(N+1)=u2(2);
         u2(N)=u2(1);
         % Update u0 and u1 for next run
         u0=u1;
         u1=u2;
         uu(i,:)=uu(i,:)+u2;
      end;
end;

% Calculate final value of uu
uu=uu/runs;
```

ZK SCHEME CODE AFTER VECTORIZATION

```
% This code uses the Zabusky-Kruskal scheme and Brownian motion to

% simulate the Stochastic KdV equation with white noise, with

% vectorization


% Due to memory allocation, we break up the code into time chunks -

% Revised


clear


% sKdV Parameters

% eta and x0 are soliton parameters

% Space interval is [a,b]

% Noise strength is given by epsilon and mu

%         epsilon is in noise correlator

%         mu is in the Gaussian white noise term


eta=1.0; x0=5.0; a=-10; b=40; epsilon = 0.1; mu=sqrt(2*epsilon);


% Run Parameters

%

% runs    = Number of realizations over the noise

% Tsteps  = Number of time steps per chunk

% NChunks = Number of time interval intervals of length Tsteps

% TTotal  = Total number of time steps

% N       = Number of spatial subintervals
% Tinc    = Increment used for saving averages

% TSave   = Number of time steps used for saving averages


runs=1; Tsteps=2000; NChunks = 8; TTotal = Tsteps*NChunks; N=500;

Tinc=10; TSave = TTotal/Tinc; cnt1=0; cnt2=0; cnt3=0;


% Time and space increments

dx=(b-a)/N; dt=dx^3/4; t=(0:TTotal-1)*dt; x=linspace(a,b,N+1);

vel=4*eta^2-4/5*eta^4*dx^2;


% Array Initializations - Grouped roughly largest to smallest

disp('Inititializing arrays ...')

uu=zeros(TSave,N+1);

uexave=zeros(TSave,N+1);

uex=zeros(Tsteps,N+1);

W=zeros(Tsteps,N+1);

u0old=zeros(runs,N+1);

u1old=zeros(runs,N+1);

totruns=zeros(size(1:TSave));

u0=zeros(size(1:N+1));

u1=zeros(size(u0));

u2=zeros(size(u0));

ichunk=zeros(size(1:Tsteps));

tchunk=zeros(size(ichunk));

w=zeros(size(ichunk));

dW=zeros(size(ichunk));

IW=zeros(size(ichunk));
```

```
dWold=zeros(size(1:runs));

wold=zeros(size(1:runs));

Iwold=zeros(size(1:runs));

ichunk2=zeros(size(1:Tsteps/Tinc));

tchunk2=zeros(size(ichunk2));

m=zeros(NChunks,1);

time=zeros(size(1:NChunks));


% uxxx and ux matrix factors using sparse matrix function spdiag

%

% A B C are for Zabusky Kruskal Scheme

% D is for noise term


N1=N+1; A=(spdiags(-2*ones(N1),1,N1,N1)+spdiags(ones(N1),2,N1,N1)

...+spdiags(2*ones(N1),-1,N1,N1)...

+spdiags(-ones(N1),-2,N1,N1))*dt/dx^3;

B=(spdiags(ones(N1),1,N1,N1)+spdiags(-ones(N1),-1,N1,N1))*dt/dx;

C=(spdiags(ones(N1),1,N1,N1)+spdiags(ones(N1),0,N1,N1)

...+spdiags(ones(N1),-1,N1,N1))*2;

D=spdiags(ones(Tsteps),0,Tsteps,Tsteps)+...

spdiags(ones(Tsteps),-1,Tsteps,Tsteps);


% Loop through time chunks

disp('Computing ...')

h = waitbar(0,'Please wait...

','Name','Zabucheck');

for chunk=1:NChunks
```

```
chunk;
t0=clock;
% The computation is broken into chunks to conserve on memory
% allocation to large matrices. Each chunk consists of Tsteps
% time steps. The interval goes from it01 to it1. All indices are
% grouped as ichunk = it01:it1 and the corresponding times are
% given as tchunk. ichunk2 is the set of indices used to saving
% the averages in steps of Tinc. tchunk2 saves the
% corresponding times.
it0=(chunk-1)*Tsteps;
it1=chunk*Tsteps;
it01=it0+1;
ichunk=it01:it1;
ichunk2=(it0/Tinc+1):(it1/Tinc);
tchunk = t(ichunk);
tchunk2(ichunk2) = t(it01:Tinc:it1);
% Loop through several ensembles
for count=1:runs

    waitbar((chunk-1+count/runs)/NChunks ,
    ...h ,['Chunk = ' num2str(chunk) ' Run = ' num2str(count)])

    % Generate Gaussian noise
    %
    % dW gives the Brownian noise increments
    % w is the Brownian noise and needs the total over increments
    % over all chinks. Thus wold saves the previous value for a
    % given run. Iw is the integral of
    % w from t=0 and is needed for the exact solution.
    % Iwold saves integral over previous chunks
    % for each run. ddW is the numerical discretization of the
    % Gaussian white needed in the sKdV. It is an average of two
    % values of neighboring increments where the 2 is suppressed
    % due to the 2dt in the forward difference
    % approximation of ut.

    dW=mu*sqrt(dt)*randn(1,Tsteps);
    w=wold(count)+cumsum(dW);
    wold(count)=w(Tsteps);
    Iw=Iwold(count)+(cumsum(w)-w/2-w(1)/2)*dt;
    Iwold(count)=Iw(Tsteps);
    ddW = D*dW';
    if chunk>1
        ddW(1)=dW(1)+dWold(count);
    end
    dWold(count)=dW(Tsteps);

    % Generate Exact Solution
    %
    % The computation of the exact solution is done using
    % vectorization and the sum of each run for all space and
    % time is saved.
    amp=2*eta^2;
    [X,T]  = meshgrid(eta*(x+x0), eta*vel*tchunk+6*eta*Iw);
```

```
W=repmat(w,N+1,1)';

uex   = W+amp*(sech(X-T)).^2;

% uex was reduced

uexave(ichunk2,:)= uexave(ichunk2,:)+uex(1:10:Tsteps,:);


% Generate Numerical Solution of PDE

%

% Initialize

% u0 = u at first time step in time block for each chunk.

% u1 = u at second time step in time block for each chunk.

% u2 = new value of u based on ZK scheme.

% uu = running sum over each run

%

% If chunk=1, then the scheme is just starting and we use the

% exact solution to obtain the first two time steps.

% For other chunks we need to carry over the old u0 and u1

% from the last chunk and same run.


if chunk==1

    u0=uex(1,:)';

    u1=u0-(A*u0)/2-((C*u0).*(B*u0))/2;

    uu(1,:)=uu(1,:)+u0';

    totruns(1)=totruns(1)+1;

    u2=zeros(size(u0));

    istart=3;

else

    u0=u0old(count,:)';
    u1=u1old(count,:)';
    istart=1;
end

% Time Loop
for i=istart:Tsteps

   % Generate solution using Zabusky-Kruskal scheme with
   % vectorization; The separate terms are vectorized

   %   UXXX=A*u1;
   %   UX=B*u1;
   %   U=C*u1;
   %   u2 = uu0-UXXX-U.*UX;

   u2 = u0 -A*u1-(C*u1).*(B*u1)+ddW(i)*ones(size(u0));

   % Calculate boundary conditions
   u2(1)=mean(u2(3:3+N/50));
   u2(2)=mean(u2(4:4+N/50));
   u2(N+1)=mean(u2(N-1-N/50:N-1));
   u2(N)=mean(u2(N-2-N/50:N-2));

   % Update u0 and u1 for next run and add u2 to
   % running sum uu totruns keeps track of
   % acceptable runs at a given time
   u0=u1;
```

```
          u1=u2;

          it0i=it0+i;

          if mod(it0i-1,Tinc)==0 & max(abs(u2))<10

                  uu(ceil(it0i/Tinc),:)=uu(ceil(it0i/Tinc),:)+u2';

                  totruns(ceil(it0i/Tinc))...

                  =totruns(ceil(it0i/Tinc))+1;

          end

       end;

       u0old(count,:)=u0;

       u1old(count,:)=u1;

   end;

   diff=etime(clock,t0);

   time(chunk)=diff;

   waitbar(chunk/NChunks,h,['Chunk = ' num2str(chunk)])

end close(h) disp('Finishing up ...') uu=uu/runs;

uexave=uexave/runs;

plot(tchunk2,max(uexave'))

hold on

plot(tchunk2,max(uu'),'r')

hold off

figure

plot(tchunk2,max(abs(uexave-uu)'))
```

ZK SCHEME TIME CHECK CODE

```
clear


% KdV Parameters
```

```
eta=1.0; x0=5.0; a=-10; b=40; epsilon = 0.1; mu=sqrt(2*epsilon);

% Run Parameters
Tsteps=2000; N=500;

% Time and space increments
dx=(b-a)/N; dt=dx^3/4; t=(0:Tsteps-1)*dt; x=linspace(a,b,N+1);
vel=4*eta^2-4/5*eta^4*dx^2;

% Generate Gaussian noise
dW=mu*sqrt(dt)*randn(1,Tsteps);
w=cumsum(dW);
Iw=(cumsum(w)-w/2-w(1)/2)*dt;

% Generate Exact Solution
amp=2*eta^2; [X,T]=meshgrid(eta*(x+x0), eta*vel*t+6*eta*Iw);
W=repmat(w,N+1,1)';
uex=W+amp*(sech(X-T)).^2;

% Generate Numerical Solution of PDE
    u0=uex(1,:);
    u1=uex(2,:);
    u2=zeros(size(u0));

    % Start clock
    t0=clock;
    % Time Loop
    for i=3:Tsteps
       ddW=(dW(i)+dW(i-1));                     % Stochastic term

       % Generate solution using Zabusky-Kruskal scheme with loops
       for j=3:N-1
          u2(j)=u0(j)-((2*dt*u1(j+1))/(dx))*(u1(j+1)-u1(j-1)+...
          u1(j)+u1(j-1)-1/(dx^2)) ...
            +((2*dt*u1(j-1))/(dx))*(u1(j)+u1(j-1)-1/(dx^2))-...
            (dt*u1(j+2))/dx^3+(dt*u1(j-2))/dx^3+ddW;
       end;

       % BCs by linear fit
       u2(1)=mean(u2(3:3+N/50));
       u2(2)=mean(u2(4:4+N/50));
       u2(N+1)=mean(u2(N-1-N/50:N-1));
       u2(N)=mean(u2(N-2-N/50:N-2));

       % Update u0 and u1 for next run
       u0=u1;
       u1=u2;
    end;

% Stop clock
t1=clock; tdiff=etime(t1,t0); disp(['Elapsed time for using loops =
', num2str(tdiff)]);
```

```
% ----------------------------------------------------------------

% Start clock
t0=clock;

 % uxxx and ux matrix factors using sparse matrix function spdiag
 N1=N+1;
 A=(spdiags(-2*ones(N1),1,N1,N1)+spdiags(ones(N1),2,N1,N1)+...
 spdiags(2*ones(N1),-1,N1,N1) ...
     +spdiags(-ones(N1),-2,N1,N1))*dt/dx^3;
 B=(spdiags(ones(N1),1,N1,N1)+spdiags(-ones(N1),-1,N1,N1))*dt/dx;
 C=(spdiags(ones(N1),1,N1,N1)+spdiags(ones(N1),0,N1,N1)+...
 spdiags(ones(N1),-1,N1,N1))*2;

 % Generate Numerical Solution of PDE
    uu0=uex(1,:)';
    uu1=uu0-(A*uu0)/2-((C*uu0).*(B*uu0))/2;
    uu2=zeros(size(uu0))';
    umax=zeros(size(t));

  % Time Loop
    for i=3:Tsteps

       % Generate solution using Zabusky-Kruskal scheme with
       % vectorization; The separate terms are

       %   UXXX=A*uu1;
```

```
        %   UX=B*uu1;
        %   U=C*uu1;                     This term contains the 6
                                         in 6u*ux
        %   uu2 = uu0-UXXX-U.*UX;

        ddW=dW(i)+dW(i-1);
        % ddW=dW(i);                    Using this shows large errors!
        uu2 = uu0 -A*uu1-(C*uu1).*(B*uu1)+ddW*ones(size(uu0));

        % BCs by linear fit
        uu2(1)=mean(uu2(3:3+N/50));
        uu2(2)=mean(uu2(4:4+N/50));
        uu2(N+1)=mean(uu2(N-1-N/50:N-1));
        uu2(N)=mean(uu2(N-2-N/50:N-2));

        % Get Error
        umax(i)=max(abs(uex(i,:)-uu2'));

        % Update u0 and u1 for next run
        uu0=uu1;
        uu1=uu2;
    end;

% Stop clock
t1=clock; tdiff=etime(t1,t0); disp(['Elapsed time without loops = ',
num2str(tdiff)]);
```

ZK SCHEME CODE WITH DAMPING

```
% This code uses the Zabusky-Kruskal scheme and Brownian motion to
% simulate the Stochastic KdV equation with white noise using
% vectorization.
clear


% sKdV Parameters
% eta and x0 are soliton parameters
% Space interval is [a,b]
% Noise strength is given by epsilon and mu
%         epsilon is in noise correlator
%         mu is in the Gaussian white noise term


eta0=1; x0=5.0; a=-10; b=90; epsilon = 0.1; gamma = 0.1;
mu=sqrt(2*epsilon);


% Run Parameters
%
% runs    = Number of realizations over the noise
% Tsteps  = Number of time steps per chunk
% NChunks = Number of time interval intervals of length Tsteps
% TTotal  = Total number of time steps
% N       = Number of spatial subintervals
% Tinc    = Increment used for saving averages
% TSave   = Number of time steps used for saving averages
```

```
runs=500; Tsteps=2000; NChunks = 16; TTotal = Tsteps*NChunks;
N=1000; Tinc=10; TSave = TTotal/Tinc; cnt1=0; cnt2=0; cnt3=0;

% Time and space increments
dx=(b-a)/N; dt=dx^3/4; t=(0:TTotal-1)*dt; x=linspace(a,b,N+1);
gamk=gamma/dx^2;

% Array Initializations - Grouped roughly largest to smallest
disp('Inititializing arrays ...')
uu=zeros(TSave,N+1);
uexave=zeros(TSave,N+1);
uex=zeros(Tsteps,N+1);
W=zeros(Tsteps,N+1);
u0old=zeros(runs,N+1);
u1old=zeros(runs,N+1);
totruns=zeros(size(1:TSave));
u0=zeros(size(1:N+1));
u1=zeros(size(u0));
u2=zeros(size(u0));
ichunk=zeros(size(1:Tsteps));
tchunk=zeros(size(ichunk));
w=zeros(size(ichunk));
dW=zeros(size(ichunk));
IW=zeros(size(ichunk));
dWold=zeros(size(1:runs));
wold=zeros(size(1:runs));
Iwold=zeros(size(1:runs));
```

```
ichunk2=zeros(size(1:Tsteps/Tinc));

tchunk2=zeros(size(ichunk2));

m=zeros(NChunks,1);

time=zeros(size(1:NChunks));


% uxxx and ux matrix factors using sparse matrix function spdiag

%

% A B C are for Zabusky Kruskal Scheme

% D is for noise term


N1=N+1;

A=(spdiags(-2*ones(N1),1,N1,N1)+spdiags(ones(N1),2,N1,N1)+...

spdiags(2*ones(N1),-1,N1,N1) ...

+spdiags(-ones(N1),-2,N1,N1))*dt/dx^3;

B=(spdiags(ones(N1),1,N1,N1)+spdiags(-ones(N1),-1,N1,N1))*dt/dx;

C=(spdiags(ones(N1),1,N1,N1)+spdiags(ones(N1),0,N1,N1)...

+spdiags(ones(N1),-1,N1,N1))*2;

D=spdiags(ones(Tsteps),0,Tsteps,Tsteps)+...

spdiags(ones(Tsteps),-1,Tsteps,Tsteps);


% Loop through time chunks

disp('Computing ...') h = waitbar(0,'Please wait...

','Name','Zabucheck'); for chunk=1:NChunks

   chunk;

   t0=clock;


   % The computation is broken into chunks to conserve on memory
   % allocation to large matrices. Each chunk consists of Tsteps
   % time steps. The interval goes from it01 to it1. All indices are
   % grouped as ichunk = it01:it1 and the corresponding times are
   % given as tchunk. ichunk2 is the set of indices
   % used to saving the averages in steps of Tinc. tchunk2
   % saves the corresponding times.

   it0=(chunk-1)*Tsteps;
   it1=chunk*Tsteps;
   it01=it0+1;
   ichunk=it01:it1;
   ichunk2=(it0/Tinc+1):(it1/Tinc);
   tchunk = t(ichunk);
   tchunk2(ichunk2) = t(it01:Tinc:it1);

   % Loop through several ensembles
   for count=1:runs

       waitbar((chunk-1+count/runs)/NChunks ,h ,...
       ['Chunk = ' num2str(chunk) ' Run = ' num2str(count)])

       % Generate Gaussian noise
       %
       % dW gives the Brownian noise increments
       % w is the Brownian noise and needs the total over increments
       % over all chunks. Thus wold saves the previous value for a
       % given run. Iw is the integral of w from t=0 and is
       % needed for the exact solution. Iwold saves integral
       % over previous chunks
       % for each run. ddW is the numerical discretization
       % of the Gaussian white needed in the sKdV. It is an
       % average of two values of neighboring increments
       % where the 2 is suppressed
       % due to the 2dt in the forward difference
       % approximation of ut.

       dW=mu*sqrt(dt)*randn(1,Tsteps);
       w=wold(count)+cumsum(dW);
       wold(count)=w(Tsteps);
       Iw=Iwold(count)+(cumsum(w)-w/2-w(1)/2)*dt;
       Iwold(count)=Iw(Tsteps);
       ddW = D*dW';
       if chunk>1
           ddW(1)=dW(1)+dWold(count);
       end
       dWold(count)=dW(Tsteps);

       % Generate Exact Solution
       %
       % The computation of the exact solution is done using
       % vectorization and the sum of each run for all space and
       % time is saved.

       [X,T]  = meshgrid(x+x0,tchunk);
```

```
W=repmat(w,N+1,1)';

uex   = 2*eta0^2*exp(-4/3*gamma*T).*(sech(eta0*...

exp(-2/3*gamma*T).*(X-3*eta0^2/gamma*...

(1-exp(-4*gamma/3*T))+3*eta0^4/10/gamk*...

(1-exp(-8*gamma/3*T))+(exp(2*...

gamma/3*T)-1)/2/eta0))).^2;


uexave(ichunk2,:)= uexave(ichunk2,:)+uex(1:10:Tsteps,:);...


% Generate Numerical Solution of PDE

%

% Initialize

% u0 = u at first time step in time block for each chunk.

% u1 = u at second time step in time block for each chunk.

% u2 = new value of u based on ZK scheme.

% uu = running sum over each run

%

% If chunk=1, then the scheme is just starting and we use the

% exact solution to obtain the first two time steps.

% For other chunks we need to carry over the old u0 and u1

% from the last chunk and same run.


if chunk==1

    u0=uex(1,:)';

    u1=u0-(A*u0)/2-((C*u0).*(B*u0))/2-gamma*u0*dt;;

    uu(1,:)=uu(1,:)+u0';

    totruns(1)=totruns(1)+1;
    u2=zeros(size(u0));
    istart=3;
else
    u0=u0old(count,:)';
    u1=u1old(count,:)';
    istart=1;
end

% Time Loop
for i=istart:Tsteps

   % Generate solution using Zabusky-Kruskal scheme with
   % vectorization; The separate terms are vectorized as

   %   UXXX=A*u1;
   %   UX=B*u1;
   %   U=C*u1;
   %   u2 = uu0-UXXX-U.*UX;

   u2 = u0 -A*u1-(C*u1).*(B*u1)+ddW(i)*ones(size(u0))-...
   gamma*u1*2*dt;

   % Calculate boundary conditions
   u2(1)=mean(u2(3:3+N/50));
   u2(2)=mean(u2(4:4+N/50));
   u2(N+1)=mean(u2(N-1-N/50:N-1));
   u2(N)=mean(u2(N-2-N/50:N-2));
```

```
          % Update u0 and u1 for next run and add u2 to
          % running sum uu totruns keeps track of
          % acceptable runs at a given time
          u0=u1;
          u1=u2;
          it0i=it0+i;
          if mod(it0i-1,Tinc)==0 & max(abs(u2))<10
                  uu(ceil(it0i/Tinc),:)=uu(ceil(it0i/Tinc),:)+u2';
                  totruns(ceil(it0i/Tinc))=...
                  totruns(ceil(it0i/Tinc))+1;
          end
       end;
       u0old(count,:)=u0;
       u1old(count,:)=u1;
   end;
   diff=etime(clock,t0);
   time(chunk)=diff;
   waitbar(chunk/NChunks,h,['Chunk = ' num2str(chunk)])
end close(h) disp('Finishing up ...') uu=uu/runs;
%plot(tchunk2,J*dx,'r');
```

BROWNIAN MOTION CHECK CODE

```
N=1000; mean=zeros(N,1); var=zeros(N,1); for j=1:N
  r=randn(N,1);
  w=zeros(N,1);
  w(1)=w0;
  for i=2:N
    w(i)=w(i-1)+sqrt(dt)*r(i);
  end
  mean=mean+w;
  var=var+w.*w;
end mean=mean/N; var=var/N;

plot(t,mean,'.',t,var,'- ');
title(['Mean and Variance of
'num2str(N) ' samples of standard Wiener Process']) xlabel('time t')
ylabel('Mean and Variance') legend('Mean','Variance',0)
```

CORRELATION CONFIRMATION CODE

```
N=1000; eps=0.1; mu=sqrt(2*eps); mean=0; for k=1:20
    s(k)=0;
    for j=1:N
        r=randn(N,1);
        w=zeros(N,1);
        w(1)=w0;
        for i=2:N
            w(i)=w(i-1)+mu*sqrt(dt)*r(i);
        end
        % Note that t=5 corresponds to i=6
        s(k)=s(k)+w(6)*w(9);
    end
    s(k)=s(k)/N;
    mean=mean+s(k);
```

```
end mean=mean/20
```

CHECK W(t) AVERAGE RELATIONSHIP CODE

```
N=200; M=1000; eps=0.1; mu=sqrt(2*eps); c=1.0; s=0; w0=0; dt=0.1;
meanw2=zeros(N,1); meanew=zeros(N,1); for i=1:M
    w2=zeros(N,1);
    ew=zeros(N,1);
    for j=1:N
        t=(j-1)*dt;
        r=randn(N,1);
        w=zeros(N,1);
        w(1)=w0;
        for i=2:N
            w(i)=w(i-1)+mu*sqrt(dt)*r(i);
        end
        w2=w.*w;
        ew=exp(c*w);
    end
    meanw2=meanw2+w2;
    meanew=meanew+ew;
end meanw2=meanw2/M; meanew=meanew/M; plot(meanw2,meanew,'-
',meanw2,exp(c^2/2*meanw2),'.')
```

PLOT SAVED DAMPING DATA CODE

```
%
clear
```

```
% Parameter Runs

G=[0.01 0.1 0.3 0.5]; E=[0.01 0.1 0.3 0.5];

fn2=['p01' 'p10''p30''p50'];

fn1='C:/Documents and Settings/Andrew Rose/My

Documents/ThesisData/skdv';

runs=[100 250 500 1000 1500];


uex=zeros(7,7,5,3200); uint=zeros(7,7,3200);


for i=1:length(runs)

    for g1=1:length(G)

        for e1=1:length(E)

            fn=[fn1, '_', fn2(3*(e1-1)+1:3*e1),'_',...

            fn2(3*(g1-1)+1:3*g1),'_',num2str(runs(i)),'.mat'];

            display([fn]);

            load(fn)

            uint(g1,e1,:)=u;

            uex(g1,e1,i,:)=u2;

        end

    end

end


% All integrals

figure(1) i1=0; for g1=1:length(G)

    for e1=1:length(E)

        i1=i1+1;

        U(1,:)=uint(g1,e1,:);
        subplot(4,4,i1); plot(t,U)
        title(['\gamma = ' num2str(G(g1)) ' \
        epsilon = ' num2str(E(e1)) ])
        axis([0,8,0,2])
    end
end

% All runs
figure(2) i1=0; for g1=1:length(G)
    for e1=1:length(E)
        i1=i1+1;
        U(1:5,:)=uex(g1,e1, 1:5,:);
        subplot(4,4,i1); plot(t,U)
        title(['\gamma = ' num2str(G(g1)) ' \
        epsilon = ' num2str(E(e1)) ])
        axis([0,8,0,2])
    end
end legend({'100','250','500','1000','1500'},...
  'Position',[0.2582 0.001209 0.5215 0.05706],...
  'Orientation','horizontal');

% Integral for fixed gamma
figure(3) for g1=1:length(G)
    U(1:7,:)=uint(g1,1:7,:);
    subplot(2,2,g1); plot(t,U)
    title(['\gamma = ' num2str(G(g1))])
    axis([0,8,0,2])
    %legend(num2str(E(1)),num2str(E(2)),...
    num2str(E(3)),num2str(E(4))...
    ,num2str(E(5)),num2str(E(6)),num2str(E(7)))
end
legend({num2str(E(1)),num2str(E(2)),num2str(E(3)),num2str(E(4))},...
  'Position',[0.2582 0.001209 0.5215 0.05706],...
  'Orientation','horizontal');

% Integral for fixed epsilon
figure(4) for e1=1:length(E)
    U(1:7,:)=uint(1:7,e1,:);
    subplot(2,2,e1); plot(t,U)
    title(['\epsilon = ' num2str(E(e1))])
    axis([0,8,0,2])
    %legend(num2str(G(1)),num2str(G(2)),num2str(G(3)),...
    num2str(G(4)),num2str(G(5)),num2str(G(6)),num2str(G(7)))
end
legend({num2str(G(1)),num2str(G(2)),num2str(G(3)),num2str(G(4))},...
  'Position',[0.2582 0.001209 0.5215 0.05706],...
  'Orientation','horizontal');



%figure(1)
%plot(t, uex(1,:),t, uex(2,:),t, uex(3,:),...
t, uex(4,:),t, uex(5,:),t ,uint(1,:))
%title('Stochastic KdV epsilon=0.1, gamma=0.1')
%xlabel('Time t')
```

```
%ylabel('Amplitude <u>')

%legend('250', '500', '1000','1500', '2000', 'Exact')

%figure(2)

%plot(t,uex(5,:),t,uint(1,:))

%title('Stochastic KdV epsilon=0.1, gamma=0.1')

%xlabel('Time t')

%ylabel('Amplitude <u>')


%Data=[t' uex' uint(1,:)'];

%dlmwrite('ExactAves_p10_p01',Data);
```

TWO SOLITON SOLUTION CODE

```
function z=twosol(x,t) p=2; q=1.5;

theta=p*x-4*p^3*(t-0.5);   % 0.5 introduced in...

 line 4 and 5 to set t=0

chi=q*x-4*q^3*(t-0.5);     % before the collision

z=2*(p^2-q^2)*((p^2+q^2*(sech(chi)).^2.*(sinh(theta)).^2)./ ...

    ((p*cosh(theta)-q*tanh(chi).*sinh(theta)).^2));

% Numerateo and denominator we multiplied by sinh(theta) to avoid

% computing csch(0) and coth(0)
```

TWO SOLITON ZK SCHEME CODE

```
% This code uses the Zabusky-Kruskal scheme and Brownian motion to

% simulate the Stochastic KdV equation with white noise for a

% 2-soliton solution.


clear
```

```
% sKdV Parameters

% eta and x0 are soliton parameters

% Space interval is [a,b]

% Noise strength is given by epsilon and mu

%         epsilon is in noise correlator

%         mu is in the Gaussian white noise term


a=-10; b=20; epsilon = 0.1; gamma = 0.01; mu=sqrt(2*epsilon);


% Run Parameters

%

% runs    = Number of realizations over the noise

% Tsteps  = Number of time steps per chunk

% NChunks = Number of time interval intervals of length Tsteps

% TTotal  = Total number of time steps

% N       = Number of spatial subintervals

% Tinc    = Increment used for saving averages

% TSave   = Number of time steps used for saving averages


runs=500; Tsteps=2000; NChunks = 4; TTotal = Tsteps*NChunks; N=400;

Tinc=10; TSave = TTotal/Tinc;


% Time and space increments

dx=(b-a)/N; dt=dx^3/4; t=(0:TTotal-1)*dt; x=linspace(a,b,N+1);

gamk=gamma/dx^2;
```

```
% Array Initializations - Grouped roughly largest to smallest
disp('Inititializing arrays ...')
uu=zeros(TSave,N+1);
%uexave = zeros(TSave,N+1);
%uex    = zeros(Tsteps,N+1);
W      = zeros(Tsteps,N+1);
u0old=zeros(runs,N+1);
u1old=zeros(runs,N+1);
totruns=zeros(size(1:TSave));
u0=zeros(size(1:N+1));
u1=zeros(size(u0));
u2=zeros(size(u0));
ichunk=zeros(size(1:Tsteps));
tchunk=zeros(size(ichunk));
w=zeros(size(ichunk));
dW=zeros(size(ichunk));
IW=zeros(size(ichunk));
dWold=zeros(size(1:runs));
wold=zeros(size(1:runs));
Iwold=zeros(size(1:runs));
ichunk2=zeros(size(1:Tsteps/Tinc));
tchunk2=zeros(size(ichunk2));
m=zeros(NChunks,1);
time=zeros(size(1:NChunks));

% uxxx and ux matrix factors using sparse matrix function spdiag
%
```

```
% A B C are for Zabusky Kruskal Scheme
% D is for noise term


N1=N+1;
A=(spdiags(-2*ones(N1),1,N1,N1)+spdiags(ones(N1),2,N1,N1)
...+spdiags(2*ones(N1),-1,N1,N1)...
    +spdiags(-ones(N1),-2,N1,N1))*dt/dx^3;
B=(spdiags(ones(N1),1,N1,N1)+spdiags(-ones(N1),-1,N1,N1))*dt/dx;
C=(spdiags(ones(N1),1,N1,N1)+spdiags(ones(N1),0,N1,N1)...
+spdiags(ones(N1),-1,N1,N1))*2;
D=spdiags(ones(Tsteps),0,Tsteps,Tsteps)+...
spdiags(ones(Tsteps),-1,Tsteps,Tsteps);


% Loop through time chunks
disp('Computing ...') h = waitbar(0,'Please wait...
','Name','Zabucheck'); for chunk=1:NChunks


   chunk;
   t0=clock;


   % The computation is broken into chunks to conserve on memory
   % allocation to large matrices. Each chunk consists of Tsteps
   % time steps. The interval goes from it01 to it1. All indices are
   % grouped as ichunk = it01:it1 and the corresponding times are
   % given as tchunk. ichunk2 is the set of indices used to saving
   % the averages in steps of Tinc. tchunk2 saves the correponding
   % times.
```

```
it0=(chunk-1)*Tsteps;
it1=chunk*Tsteps;
it01=it0+1;
ichunk=it01:it1;
ichunk2=(it0/Tinc+1):(it1/Tinc);
tchunk = t(ichunk);
tchunk2(ichunk2) = t(it01:Tinc:it1);

% Loop through several ensembles
for count=1:runs

    waitbar((chunk-1+count/runs)/NChunks ,h ,...
    ['Chunk = ' num2str(chunk) ' Run = ' num2str(count)])

    % Generate Gaussian noise
    %
    % dW gives the Brownian noise increments
    % w is the Brownian noise and needs the total over increments
    % over all chinks. Thus wold saves the previous value for a
    % given run. Iw is the integral of w from t=0 and is needed
    % for the exact solution. Iwold saves integral over previous
    % chunks for each run. ddW is the numerical discretization of
    % the Gaussian white needed in the sKdV. It is an average of
    % two values of neighboring increments where the 2 is
    % suppressed due to the 2dt in the forward difference
    % approximation of ut.
```

```
dW=mu*sqrt(dt)*randn(1,Tsteps);

w=wold(count)+cumsum(dW);

wold(count)=w(Tsteps);

Iw=Iwold(count)+(cumsum(w)-w/2-w(1)/2)*dt;

Iwold(count)=Iw(Tsteps);

ddW = D*dW';

if chunk>1

    ddW(1)=dW(1)+dWold(count);

end

dWold(count)=dW(Tsteps);


% Generate Numerical Solution of PDE

%

% Initialize

% u0 = u at first time step in time block for each chunk.

% u1 = u at second time step in time block for each chunk.

% u2 = new value of u based on ZK scheme.

% uu = running sum over each run

%

% If chunk=1, then the scheme is just starting and we use the

% exact solution to obtain the first two time steps.

% For other chunks we need to carry over the old u0 and u1

% from the last chunk and same run.


if chunk==1

    u0=twosol(x,0)';
    u1=u0-(A*u0)/2-((C*u0).*(B*u0))/2-gamma*u0*dt;;
    uu(1,:)=uu(1,:)+u0';
    totruns(1)=totruns(1)+1;
    u2=zeros(size(u0));
    istart=3;
else
    u0=u0old(count,:)';
    u1=u1old(count,:)';
    istart=1;
end

% Time Loop
for i=istart:Tsteps

   u2 = u0 -A*u1-(C*u1).*(B*u1)+ddW(i)*ones(size(u0))-...
   gamma*u1*2*dt;

   % Calculate boundary conditions
   u2(1)=mean(u2(3:3+N/50));
   u2(2)=mean(u2(4:4+N/50));
   u2(N+1)=mean(u2(N-1-N/50:N-1));
   u2(N)=mean(u2(N-2-N/50:N-2));

   % Update u0 and u1 for next run and add
   % u2 to running sum uu
   % totruns keeps track of acceptable runs at a given time
   u0=u1;
```

```
            u1=u2;
            it0i=it0+i;
            if mod(it0i-1,Tinc)==0 & max(abs(u2))<10
                    uu(ceil(it0i/Tinc),:)=...
                    uu(ceil(it0i/Tinc),:)+u2';
                    totruns(ceil(it0i/Tinc))=...
                    totruns(ceil(it0i/Tinc))+1;
            end
        end;
        u0old(count,:)=u0;
        u1old(count,:)=u1;
    end;
    diff=etime(clock,t0);
    time(chunk)=diff;
    waitbar(chunk/NChunks,h,['Chunk = ' num2str(chunk)])
end close(h) disp('Finishing up ...') uu=uu/runs; mesh(x,tchunk2,uu)
view(-30,60)

for i=1:length(tchunk2)
     [aa,bb]=lmax(uu(i,:),2);
     [c,ic]=sort(aa,'descend');
     amp1(i)=aa(ic(1));
     pos1(i)=bb(ic(1));
     amp2(i)=aa(ic(2));
     pos2(i)=bb(ic(2));
end
```

```
figure(1) [X,T]=meshgrid(x,tchunk2); U=twosol(X,T);

for i=1:length(tchunk2)
    [a,b]=lmax(U(i,:),2);
    if length(a)==1
        a(2)=a(1);
        b(2)=b(1);
    end
    if a(1)>a(2)
        amp3(i)=a(1);
        pos3(i)=b(1);
        amp4(i)=a(2);
        pos4(i)=b(2);
    else
        amp3(i)=a(2);
        pos3(i)=b(2);
        amp4(i)=a(1);
        pos4(i)=b(1);
    end
end figure(2) plot(tchunk2,x(pos1),tchunk2,x(pos2)) hold on
plot(tchunk2,x(pos3),'k',tchunk2,x(pos4),'r') title('Position vs
Time') hold off

figure(3) plot(tchunk2,amp1,tchunk2,amp2) hold on
plot(tchunk2,amp3,'k',tchunk2,amp4,'r') title('Amplitude vs Time')
hold off
```

TWO SOLITON PLOT CODE

```
clear x=(-10:.1:10); t=(0:.01:1); [X,T]=meshgrid(x,t);

U=twosol(X,T);


figure(1) mesh(x,t,U); title('Two Soliton Solution of KdV')

xlabel('x') ylabel('t') zlabel('u(x,t)') view(-30,60)

% view(0,90)   % Shows phase shift


figure(2) for i=1:length(t)

    [a,b]=lmax(U(i,:),2);

    if a(1)>a(2)

        amp1(i)=a(1);

        pos1(i)=b(1);

        amp2(i)=a(2);

        pos2(i)=b(2);

    else

        amp1(i)=a(2);

        pos1(i)=b(2);

        amp2(i)=a(1);

        pos2(i)=b(1);

    end

end plot(t,x(pos1),t,x(pos2)) title('Position vs Time')


figure(3) plot(t,amp1,t,amp2) title('Amplitude vs Time')
```

SUPPLEMENTARY FUNCTIONS

```
%Exact solution function
```

```
function z=func(x,t,eta,x0)

z=2*eta^2*(sech(eta*(x-4*eta^2*t+x0))).^2;


%Plot Linear fit for log-log plot and find rho

z=max(uu'); plot(log(t),log(z))

p=polyfit(log(t(100:400)),log(z(100:400)),1)

plot(log(t),log(z),'.',log(t),polyval(p,log(t))

,'-','MarkerSize',20,'LineWidth',2)

x=t(100:400);

y=z(100:400);

cc=corrcoef(x,y);

rho=cc(1,2)

rho^2


%Find confidence interval

(abs(p)*tan(arccos(rho^2)))/((N-2)^(1/2))
```

BIOGRAPHICAL SKETCH

This thesis is being submitted as a requirement for Andrew Rose to receive his Master of Science in Mathematics. Prior to his course work in mathematics, he received his Bachelor of Science in Computer Science from UNCW (December 2003). He has worked in ITSD of UNCW for 5 years and has been a TA in the math department for 2 years. He hopes to continue to work with computers and math in the future.