5-2019

A Qubit Algorithm for Simulating the Nonlinear Schroedinger Equation

Connor Simpson

A Qubit Algorithm for Simulating the Nonlinear Schroedinger Equation

A thesis submitted in partial fulfillment of the requirement for the degree of Bachelor of Science with Honors in Physics from the College of William and Mary in Virginia,

by

Connor N. Simpson

Accepted for Honors
(Honors or no-Honors

[illegible]
Advisor: Prof. George Vahala

[illegible]
Prof. Seth Aubin

[illegible]
Prof. Matthew Haug

Williamsburg, Virginia
May 3, 2019

Contents

Acknowledgments

I would like to thank my advisor, Dr. Vahala, for his support and encouragement, as well as my peers for their advice during the course of my project.

List of Figures

Abstract

Recent work in mathematical physics and nonlinear optics has shown that Hamiltonians that are non-Hermitian but still symmetric under parity and time reversal can describe eigenstates of a system with real eigenvalues. Other research has also showed that the nonlinear Schrodinger equation can be generalized to describe PT-symmetric systems, which generates novel solutions not described by its Hermitian equivalent. The Hermitian form of the nonlinear Schroedinger equation can also be extended to describe a particular case of the general PT-symmetric NLS, suggesting a connection between the two. I attempted to generate a unitary operator that will be useful for unitary quantum algorithms describing a coupled set of nonlinear Schroedinger equations and the PT-symmetric version of the NLS.

Chapter 1

Introduction

1.1 PT Symmetry

Traditionally, a nonrelativistic quantum system is described by a Hamiltonian $\hat{H}$ of the form

$$\hat{H} = \frac{\hat{p}^2}{2m} + \hat{V}_{ext}(\vec{r}) + \sum_{i,j} \hat{V}_{ij} \tag{1.1}$$

where $\hat{p}$ is the momentum operator, $\hat{V}_{ext}$ is the external potential, and the sum of $\hat{V}_{ij}$ is the interaction term. In order for this operator to give orthogonal eigenstates with real eigenvalues corresponding to the energy eigenstates of the physical system, it has usually been assumed that $\hat{H}$ be Hermitian. That is,

$$\hat{H} = \hat{H}^\dagger \tag{1.2}$$

where $\dagger$ is the ordinary complex conjugate transpose.

This is a sufficient requirement for the eigenstates of the operator to have real eigenvalues and thus for it to describe quantum mechanics. However, the question remains as to whether this is a necessary requirement. According to [3] we may also impose a requirement that encompasses more scenarios than Hermiticity on the Hamiltonian which still ensures that all of the system's eigenvalues are real, which is that it is symmetric under the operation PT, where P is the parity operator and T

is the time reversal operator, defined as

$$P : x \to -x, p \to -p \tag{1.3}$$

$$T : t \to -t, i \to -i \tag{1.4}$$

where x, t, and p are position, momentum, and time, and i is the square root of -1. Note that the time reversal operator must include the complex conjugate in order to preserve the commutator $[\hat{x}, \hat{p}] = i\hbar$. It has been shown in [3] that all Hamiltonians with unbroken PT symmetry, that is $[H, PT] = 0$, possess a real spectrum of eigenvalues and the eigenfunctions of H are also eigenfunctions of PT. Of interest to us is the PT-symmetric version of the nonlinear Schroedinger equation, which I will discuss in detail in the next section.

1.2 The Nonlinear Schroedinger Equation

A diverse set of both classical and quantum systems will yield an equation of the form

$$i\frac{\partial \psi}{\partial t} = -\frac{1}{2}\frac{\partial^2 \psi}{\partial x^2} \pm g|\psi|^2\psi \tag{1.5}$$

Where ψ is a complex scalar field whose physical significance depends on context. For example, in a Bose-Einstein condensate this equation describes the time evolution of the aggregate wave function, which is also the order parameter associated with phase transition. This equation also appears in nonlinear optics, where it describes the propagation of optical polarizations through a nonlinear medium such as a fiber optic cable. The NLS also describes the envelope of waves propagating in shallow water. What is of interest about the NLS is there exist solutions known as solitons that propagate at a constant velocity without dispersion. In other words, they will always maintain their shape no matter how far in time they have propagated. This makes them perfect candidate solutions for testing the efficacy of algorithms that solve nonlinear differential equations. When the sign of the potential $|\psi|^2$ is positive, the solitons will have positive amplitudes and be known as bright solitons, while they will appear as depressions in a constant, nonzero scalar field and be known as dark solitons if the potential is negative. For convenience, I will set $\hbar = m = 1$ for all of the equations we will be looking at.

The NLS admits several solutions. If we impose the boundary condition that $|\psi| \to 0$ as $x \to \infty$, what we get is a soliton travelling with some velocity v described by the equation

$$\psi(x,t) = e^{(\mu t + vx/2)}\sqrt{\mu} sech[\sqrt{\mu}(x - vt)] \tag{1.6}$$

where v is the group velocity of the soliton and μ is a normalization factor that affects both the amplitude and phase of the soliton. The dynamics of solitons are well-known, and I will not discuss them here. The important lesson to learn from this particular case is that solutions of the NLS must conform to a particular length-amplitude scaling. This will become relevant later.

A more interesting solution to the NLS and one that has not been simulated in the qubit regime is the Peregrine solution described in [5], which is a soliton whose amplitude oscillates in time. This solution appears when we impose the boundary condition that $|\psi|$ approaches some constant, nonzero value as x approaches infinity and is described by the function

$$\psi(x,t) = e^{2it}\frac{cos(\Omega t - 2i\phi) - cosh(\phi)cosh(px)}{cos(\Omega t) - cosh(\phi)cosh(px)} \tag{1.7}$$

where ϕ is the period of the soliton, $\Omega = 2sinh(2\phi)$ and $p = 2sinh(\phi)$. If the period of the breather is taken to infinity, then we get a solution described by the equation

$$\psi(x,t) = 1 - \frac{4(1+4it)}{1+4x^2+16t^2} \tag{1.8}$$

Surprisingly, this function peaks only once, at $t = x = 0$. This behavior is similar to the observed behavior of so-called "rogue waves" in the ocean, and suggests the existence of similar phenomena in other systems governed by the NLS.

1.3 The Coupled NLS

I will primarily be observing the theoretical behavior of solitons governed by two sets of equations. One is a set of coupled equations that govern the dynamics of two coupled optical waveguides, which can be approximated as

$$iu_t + u_{xx} + 2|u|^2u = -v + i\gamma u \tag{1.9}$$

$$iv_t + v_{xx} + 2|v|^2v = -u - i\gamma v \tag{1.10}$$

where u and v are the polarizations of the optical field and each subscript is a derivative with respect to the variable in question. These coupled equations admit breather solutions. Because these equations are non-Hermitian, there is no guarantee that a unitary algorithm will properly simulate them. The first method for getting around this is provided by Barashenkov in [2]. He makes two successive transforms. The first is to do a simple rotation of coordinates:

$$a = \frac{e^{i\theta}u - v}{2\omega_0} \tag{1.11}$$

$$b = \frac{e^{-i\theta}u + v}{2\omega_0} \tag{1.12}$$

where

$$\theta = arcsin\gamma \tag{1.13}$$

$$\omega_0 = cos\theta \tag{1.14}$$

To start the multi-timescale expansion, he makes an additional substitution:

$$a(x,t) = \epsilon^{1/2}A(x,t) \tag{1.15}$$

$$b(x,t) = \epsilon^{1/2}B(x,t) \tag{1.16}$$

If we make these two substitutions, we end up with the equations for A and B:

$$iA_t + \epsilon A_{XX} - \omega_0 A + 2\epsilon(|A|^2 + 2|B|^2)A + 4ie^{-i\theta}\epsilon\gamma A^2 B^* + 2e^{2i\theta}\epsilon A^* B^2 = 0 \quad (1.17)$$

$$iB_t + \epsilon B_{XX} + \omega_0 A + 2\epsilon(2|A|^2 + |B|^2)B - 4ie^{i\theta}\epsilon\gamma B^2 A^* + 2e^{-2i\theta}\epsilon B^* A^2 = 0 \quad (1.18)$$

The basic method for generating the equations we solve for the initial conditions is to define new coordinates and derivatives:

$$T_n = \epsilon^n t \quad (1.19)$$

$$X_n = \epsilon^n X \quad (1.20)$$

$$D_n = \partial/\partial T_n \quad (1.21)$$

$$\partial_n = \partial/\partial X_n \quad (1.22)$$

We can also expand A and B into different scales such that $A = A_0 + \epsilon A_1 + ...$ and $B = B_0 + \epsilon B_1 + ...$ This allows us to likewise expand the equations above into equations for the A and B terms of each order. The zeroth order equation is:

$$(iD_0 - w_0)A_0 = 0 \quad (1.23)$$

$$(iD_0 + w_0)B_0 = 0 \quad (1.24)$$

which leads to the obvious solutions $A_0 = e^{-iw_0 T_0}p$ and $B_0 = e^{iw_0 T_0}q$. We can then expand out to first order and plug these in to get the first order equations, and then take the solutions to that to get the second order equations. At all orders, we observe secular terms that break the ordering, which we set to zero. Adding them together gives the approximate behavior of the system at orders up to ϵ^2. The equations for p and q are

$$ip_T + p_{XX} + 2(|p|^2 + 2|q|^2)p + \frac{\epsilon}{\omega_0}(|q|^2 - 2|p|^2)|q|^2 p = 0 \quad (1.25)$$

$$iq_T + q_{XX} + 2(|q^2 + 2|p|^2)q + \frac{\epsilon}{\omega_0}(2|q|^2 - |p|^2)|p|^2 q = 0 \quad (1.26)$$

One question we might ask is whether these equations admit breather solutions. If they do, then said solutions will take the form

$$p = e^{i\mu T} P(X) \tag{1.27}$$

$$q = e^{i\nu T} Q(X) \tag{1.28}$$

Plugging these into the original equations for p and q and setting $\mu = \nu$ gives partial solutions. To get the solutions we want we need to further expand P and Q to accommodate for the previous time scale expansion. We eventually get solutions for p and q of

$$p = \frac{e^{iT}}{\sqrt{3}} sechX[1 - \frac{\epsilon}{102\omega_0}(6 + sech^2 X) + O(\epsilon^2)] \tag{1.29}$$

$$q = \frac{e^{iT}}{\sqrt{3}} sechX[1 + \frac{\epsilon}{102\omega_0}(6 + sech^2 X) + O(\epsilon^2)] \tag{1.30}$$

which can be converted back to u and v via inverting our original transformations.

Chapter 2

The Qubit Algorithm

To create a method of simulating the behavior of the types of nonlinear Schroedinger equation I described above on a quantum computer, I will construct a unitary operator that corresponds to the time evolution operator of the general system. I will test the accuracy of this operator by applying it to known solutions to both equations, i.e. the breather solitons discussed in the last section. I will construct operators that simulate both the coupled and uncoupled equations, the latter of which is simply a degenerate case of the former.

2.1 Qubits

The basic building blocks of any quantum algorithm are qubits. A qubit is a quantum object whose wave function ϕ can be represented as a superposition of two states $|0>$ and $|1>$ such that

$$\phi = \alpha|0> + \beta|1> \tag{2.1}$$

where α and β are both complex numbers. Note that when a qubit is observed it will be observed as in either the $|0>$ or $|1>$ state, both of which correspond to the respective 0 and 1 states of a classical bit. The advantage of qubits is that two can be entangled to represent systems that cannot be described by two classical bits. For example, the state

$$\alpha|00> + \beta|11> \tag{2.2}$$

cannot be expressed as a tensor product of two independent qubit states and thus cannot be generated on a classical computer. By applying a sequence of operators to a large number of entangled qubits, we can construct circuits that perform various computations, including the simulation of quantum systems that cannot be directly simulated on a classical computer.

2.2 The General Method

To begin, consider a generic scalar field governed by the *linear* one-dimensional Schroedinger equation

$$i\frac{\partial \psi}{\partial t} = \hat{H}\psi \tag{2.3}$$

where $\hat{H}$ is the Hamiltonian $-\frac{1}{2}\frac{\partial^2\psi}{\partial x^2} + V(x)$ and $V(x)$ is some external scalar potential. The general time dependent solution to this equation, given some initial wave function ψ_0, is

$$\psi(x,t) = e^{-i\hat{H}t}\psi_0(x) \tag{2.4}$$

To simulate this on a classical computer, we would apply the unitary time evolution operator $e^{-i\hat{H}\Delta t}$ in increments of Δt to ψ_0 and observe its evolution. However, if we want to simulate this system on a quantum computer we will need a slightly different method. First, we break up the wavefunction into two separate quantities, q_0 and q_1. We then note that for each time step

$$q_0(x, t + \Delta t) = aq_0(x,t) + bq_1(x,t) \tag{2.5}$$

$$q_1(x, t + \Delta t) = cq_0(x,t) + dq_1(x,t) \tag{2.6}$$

where a, b, c, and d are complex numbers. This set of equations and thus the time evolution operator of this system can be represented by the matrix

$$U(x,t) = \begin{pmatrix} a(x,t) & b(x,t) \\ c(x,t) & d(x,t) \end{pmatrix} \tag{2.7}$$

which represents the time evolution of the system over some time Δt. If the Hamiltonian of the system in question is Hermitian and/or PT symmetric, this matrix must be unitary. Thus, the problem becomes one of finding a matrix $U(x,t)$ that is unitary such that $U(q_0(x,t), q_1(x,t)) = (q_0(x,t+\Delta t), q_1(x,t+\Delta t)) + \mathcal{O}(\epsilon^3)$.

2.3 The Nonlinear Schroedinger Equation

The qubit case of the discrete time evolution operator for the scalar field equation expressed in equation 1.5 is

$$U = e^{-i\partial_x^2 \Delta t/2} e^{-i|q_0+q_1|^2 \Delta t} \tag{2.8}$$

As seen above, the operator is split into two parts: the kinetic energy part, and the potential energy part. It is notable that these operators do not commute, and thus we must interleave them together when actually applying the algorithm. As shown in [1], we see that interleaving the components of this operator properly is enough for this to approximate the evolution of the system to within second order. First, consider the unitary collision operator C that locally entangles q_0 and q_1

$$\sqrt{SWAP} = \begin{pmatrix} \frac{1-i}{2} & \frac{1+i}{2} \\ \frac{1+i}{2} & \frac{1-i}{2} \end{pmatrix} \tag{2.9}$$

and the streaming operators

$$S_{\Delta x,0} = n_+ + e^{\Delta x \partial_x} n_- \tag{2.10}$$

$$S_{\Delta x,1} = n_- + e^{\Delta x \partial_x} n_+ \tag{2.11}$$

where $n_\pm = (1 \pm \sigma_z)/2$, which shift q_0 and q_1 by a length Δx. To represent the movement of qubits on a lattice, we construct the operator $J_{x\gamma} = S_{-\Delta x,\gamma} C S_{\Delta x,\gamma} C$. The operator $J_{x\gamma}^2$ then represents the total kinetic energy operator acting on the γth qubit.

Adding in the potential, we get a total operator

$$J_{x0}^2 e^{-i\epsilon^2 \Omega(x)/2} J_{x1}^2 e^{-i\epsilon^2 \Omega(x)/2} \tag{2.12}$$

where

$$\Omega(x) = \begin{pmatrix} cos[\Omega \Delta t] & -isin[\Omega \Delta t] \\ -isin[\Omega \Delta t] & cos[\Omega \Delta t] \end{pmatrix} \tag{2.13}$$

Where Ω is the scalar potential we're working with. It can easily be shown that this is the operator that approximates the unitary time evolution operator to within order ϵ^3. In the NLS case, the potential is $|\psi|^2$. While C is suitable for handling low-amplitude solutions to the NLS, research has shown that a collision operator based on the Dirac equation is a better fit for simulating the NLS, which is

$$C_D = \begin{pmatrix} cos(\frac{pi}{4} - \frac{1}{8}|\psi|^2) & -isin(\frac{pi}{4} - \frac{1}{8}|\psi|^2) \\ -isin(\frac{pi}{4} - \frac{1}{8}|\psi|^2) & cos(\frac{pi}{4} - \frac{1}{8}|\psi|^2) \end{pmatrix} \tag{2.14}$$

This collision operator replaces both the original collision operator C and the the perturbative terms $e^{-i\epsilon^2 \Omega(x)/2}$ and $e^{-i\epsilon^2 \Omega(x)/2}$ in the old algorithm, giving us fidelity at much higher amplitudes.

2.4 Rescaling

The qubit algorithm presented in the last two sections is perturbative. Hence, we expect nonphysical behavior of high-amplitude solutions that must be corrected for. However, due to the nonlinearity of the Manakov equations, this is not a simple matter of multiplying the amplitude by some factor α. Instead, we also have to multiply x by some other factor $1/\chi$, where χ is the characteristic length scale of the function.

Now, we must also multiply t by some characteristic time scale $1/\tau$, but due to the diffusion ordering of the algorithm we know that $1/\chi = 1/\tau^2$. If we consider some function $\alpha phi(X, T)$ where $X = x/\chi$ and $T = t/\tau$ and plug it into the NLS equation, we get

$$i\alpha\beta^2\phi_t + \alpha\beta^2\phi_{xx} + 2\alpha^3|\phi|^2\phi = 0 \tag{2.15}$$

where $\beta = 1/\chi$. If we make the additional assumption that $\phi(X, T)$ is still a solution of the NLS equation, then the only way for this expression to hold is if $\alpha = \pm\beta$. Our final result is therefore that if $phi(x, t)$ is a solution then so is $\alpha\phi(\alpha x, \alpha^2 t)$.

Chapter 3

Numerical Calculations

To test the algorithm, I used code written in Fortran that evolves a starting wavefunction using the qubit operator detailed in the last section. The code first instantiates the wavefunction on a grid with 1024 lattice points and then breaks it into the q_0 and q_1 qubits. The code applies the Dirac version of the unitary qubit operator to both qubits on every lattice point at each time step. At certain designated time steps, the code also plots the sum of the two qubits, which corresponds to the scalar field we want to recover.

For the algorithm to accurately the equation, it must obey several conservation laws. These include conservation of unitarity and conservation of energy, which the coupled NLS can be shown to obey. Thus, in addition to checking the scalar field at regular intervals, the code also checks the total probability current of the scalar field and the potential plus the kinetic energy of the system to ensure that the conservation laws hold. In addition to unitarity, the code must also maintain periodic boundary conditions throughout the evolution of the system. To do this, the code uses a circular array shift every time the first and last points on the lattice grid change.

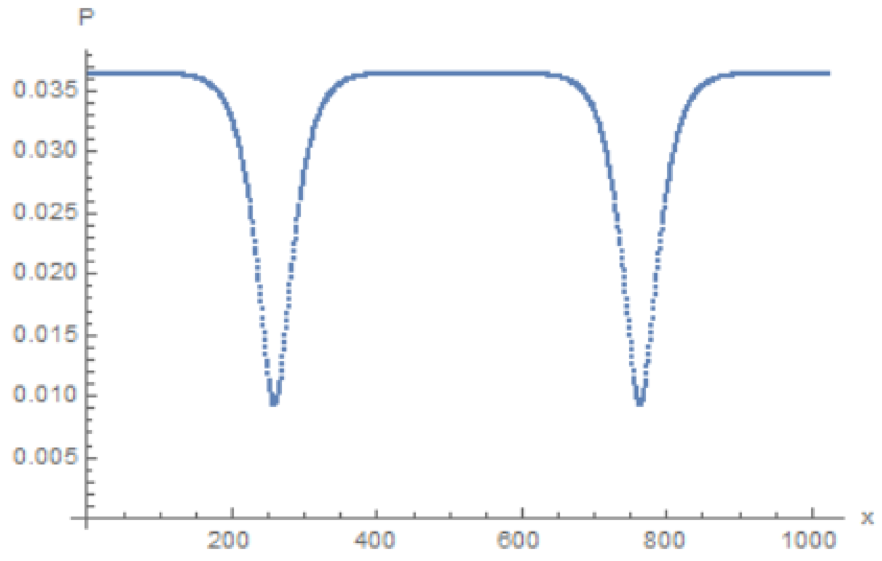

Figure 3.1: Absolute value versus position of two solitons with an amplitude of .0125/2.59075 at $t_0 = 0au$ colliding at a velocity of .00125. At this speed they do not exhibit any noise.

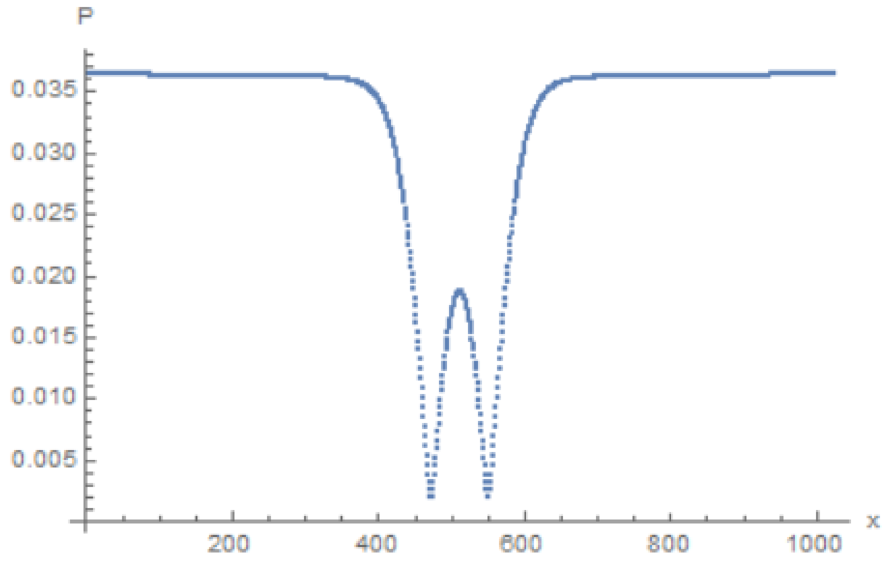

Figure 3.2: Absolute value versus position of two solitons with an amplitude of .0125/2.59075 at $t_1 = 800au$.

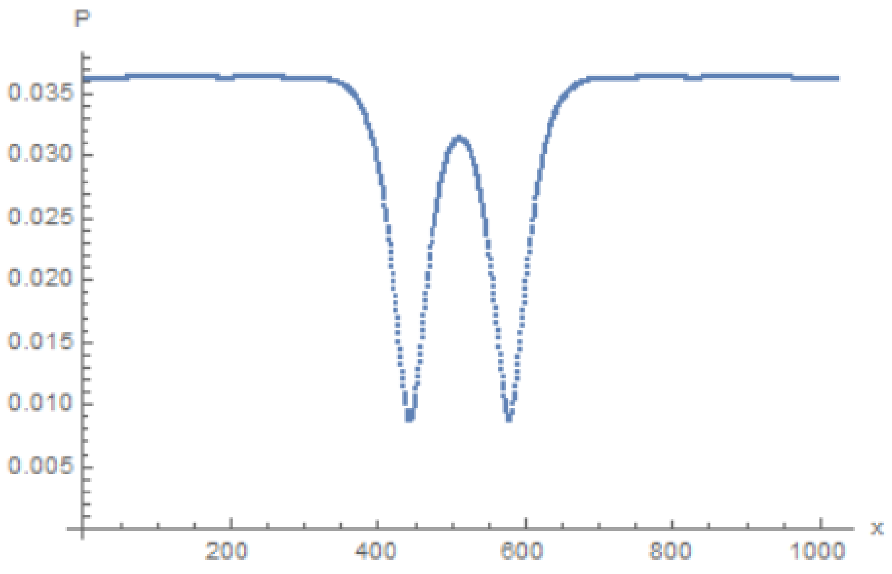

Figure 3.3: Absolute value versus position of two solitons with an amplitude of .0125/2.59075 at $t_2 = 1000au$.

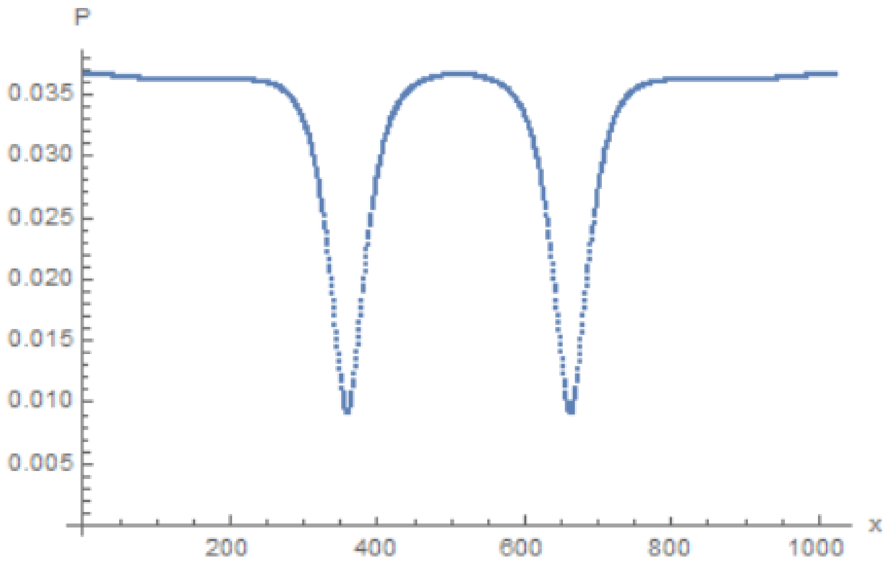

Figure 3.4: Absolute value versus position of two solitons with an amplitude of .0125/2.59075 at $t_3 = 1300au$.

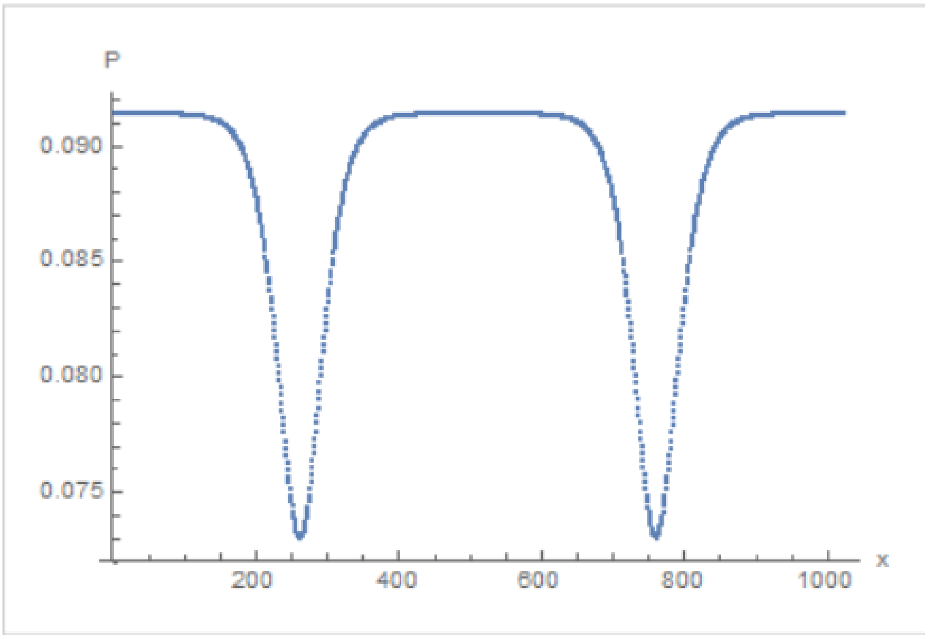

Figure 3.5: Absolute value versus position of two solitons with an amplitude of .0125/2.59075 colliding at a velocity of .125 at $t_0 = 0au$. At this speed they exhibit noise in the form of higher frequency oscillations, which is an artifact of the algorithm.

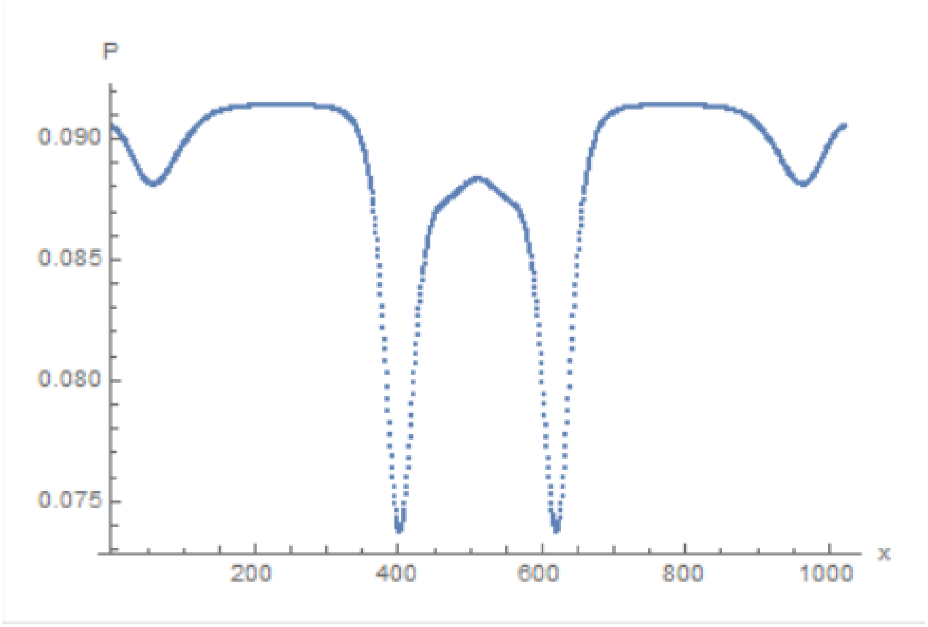

Figure 3.6: Absolute value versus position of two solitons with an amplitude of .0125/2.59075 colliding at a velocity of .125 at $t_1 = 500au$.

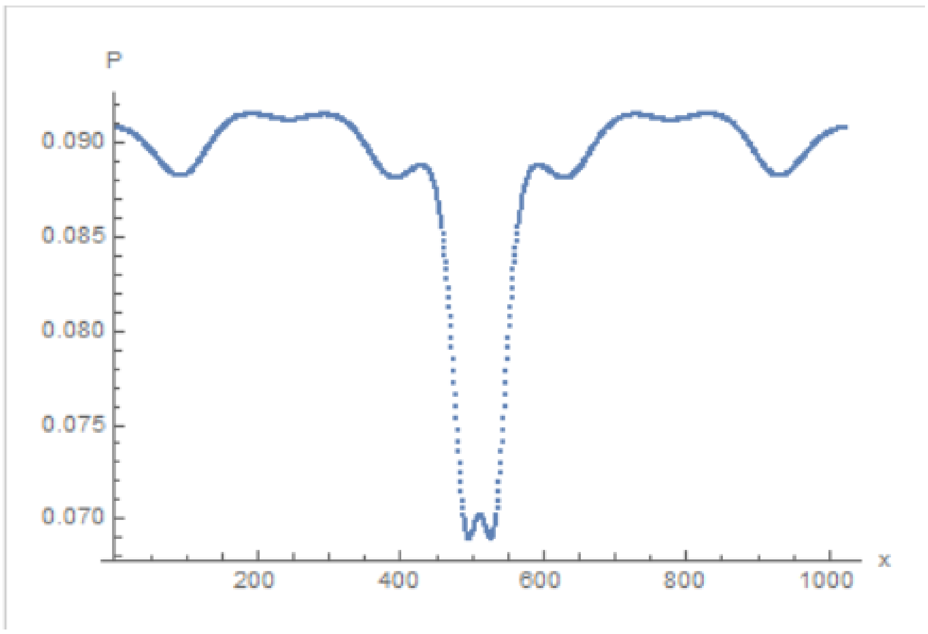

Figure 3.7: Absolute value versus position of two solitons with an amplitude of .0125/2.59075 colliding at a velocity of .125 at $t_2 = 1000au$.

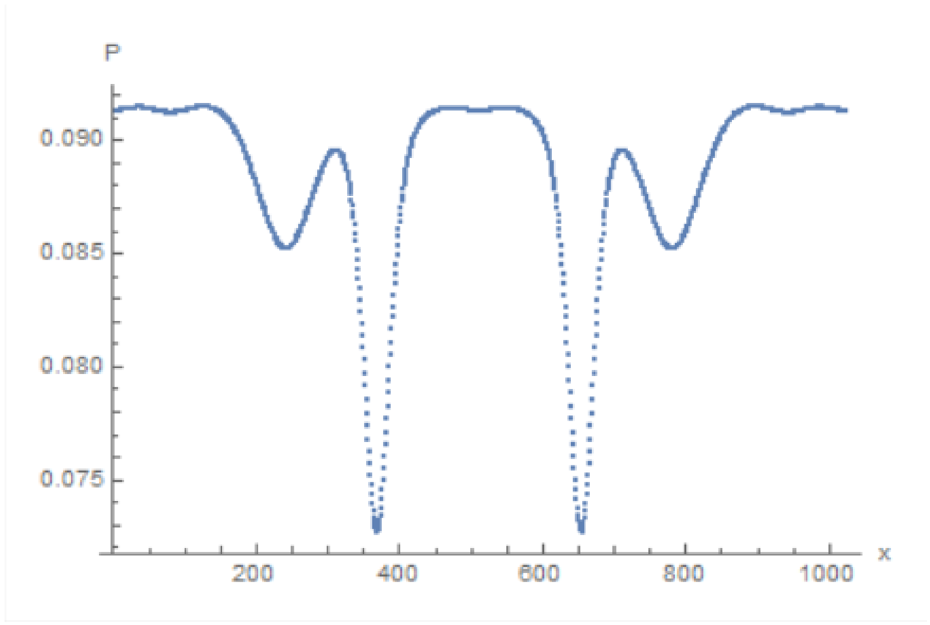

Figure 3.8: Absolute value versus position of two solitons with an amplitude of .0125/2.59075 colliding at a velocity of .125 at $t_3 = 1300$.

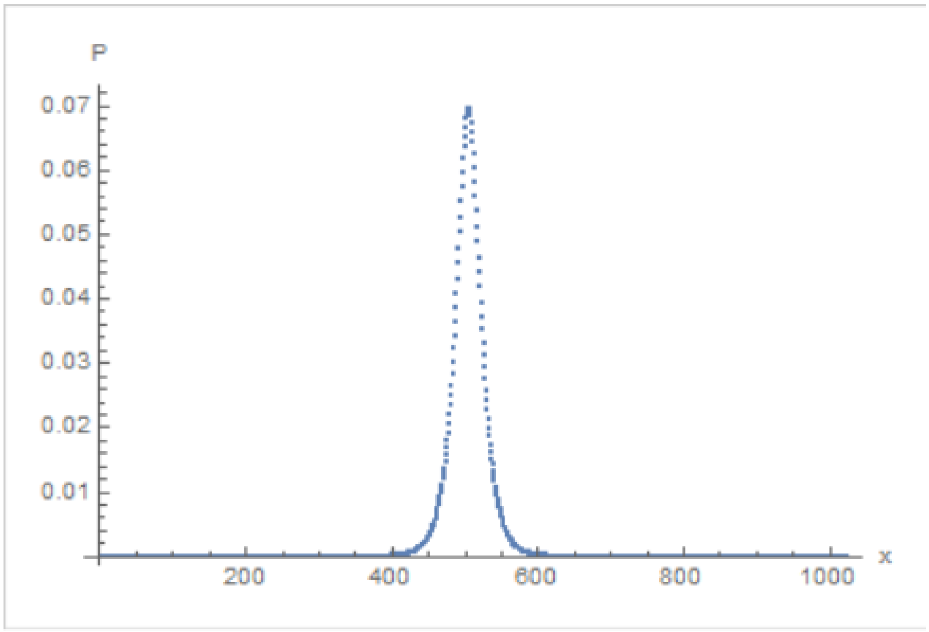

Figure 3.9: Absolute value versus position of the degenerate soliton solution to the Manakov system at $t_0 = 0au$. In this regime, the system is well-behaved.

3.1 Data Collection

So far, I have taken plots of the collision of two dark solitons governed by equation 1.6. The solitons can be observed to move at a constant velocity and at low velocities the collision does not cause any interference, as seen in figures 3.1-3.4. However, at higher velocities the algorithm becomes unstable, specifically around more than one grid space per time step, as in figures 3.5-3.8. Despite this instability, the algorithm always remains unitary.

I also simulated the coupled Manakov system detailed in equations 6 and 7 using the same general qubit method. I first tried a degenerate bright soliton with $q = 0$ solution moving at a constant velocity as shown in figures 3.9-3.11. I found that unitarity was conserved in this case. I then checked the case of an inelastic collision and found that minor solitons travelling in the opposite direction were formed dur-

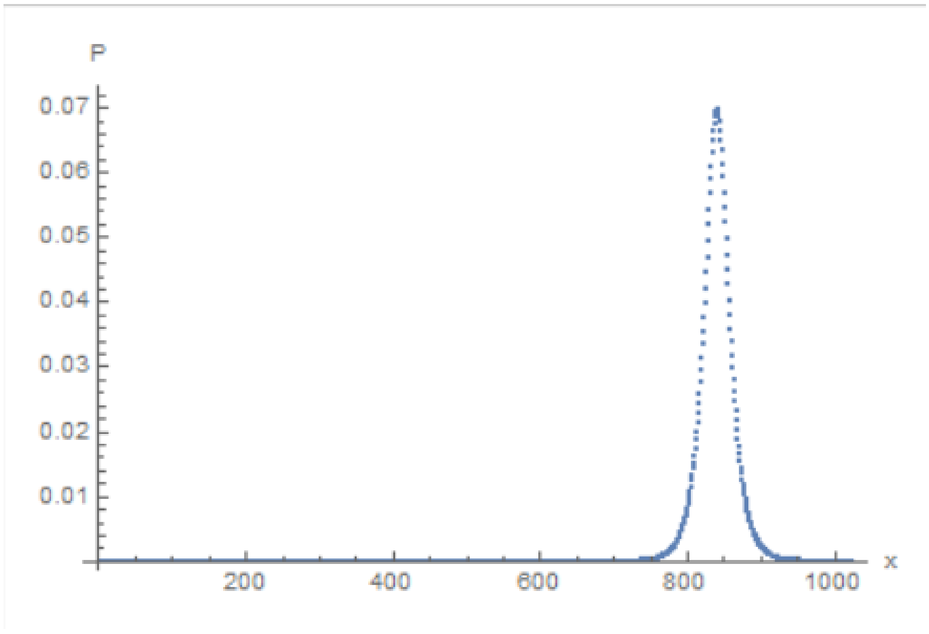

Figure 3.10: Absolute value versus position of the degenerate soliton solution to the Manakov system at $t_1 = 500au$.

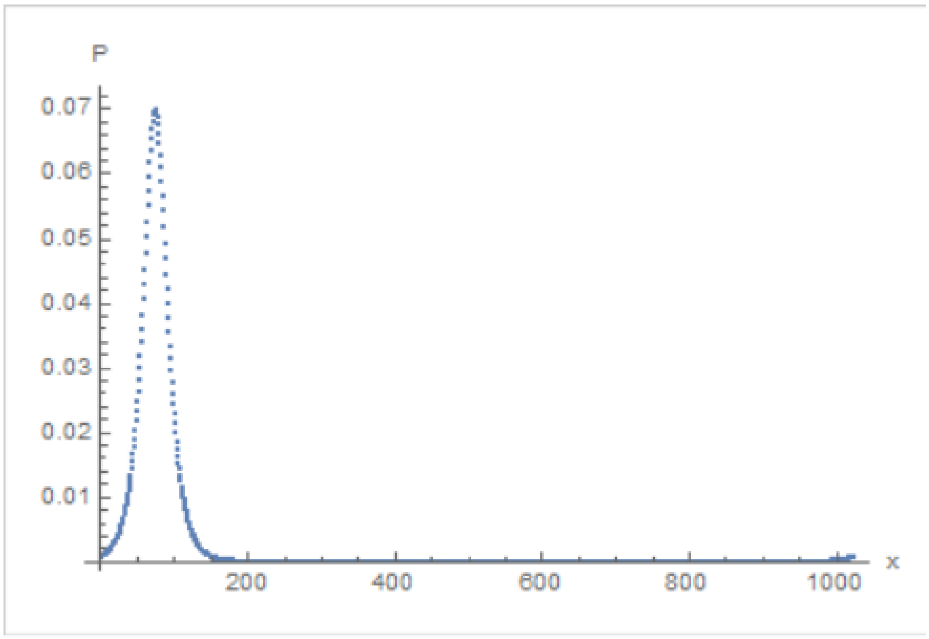

Figure 3.11: Absolute value versus position of the degenerate soliton solution to the Manakov system at $t_2 = 1000au$.

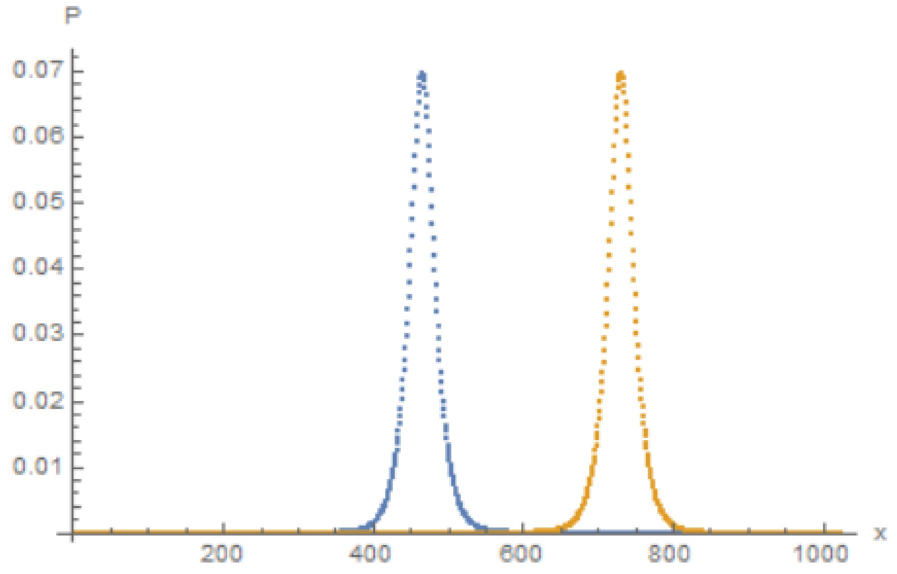

Figure 3.12: Absolute value versus position of the inelastic collision solution to the Manakov system, simulated on a 1024-qubit grid at $t_0 = 0au$. The orange wave function is the norm squared of the q field, while the blue wave function is the norm squared of the p field. The formation of extra solitons is due to the coupled system having a degree of freedom with respect to the number of solitons that can be present before and after collision so long as energy is conserved.

ing collision, particularly in the p wavefunction, as seen in figures 3.12-3.14. The Peregrine solution to the NLS presents us with an interesting challenge. Rather than simulating the original solution presented in equation 1.7, I rescaled it using a scaling coefficient $\alpha = \frac{1}{200}$. The Peregrine solution is expected to diminish to a constant function, but in my simulation the function dips into what resembles a broadened dark soliton, as seen in figures 3.15-3.20. The most likely explanation for this is that there is something wrong with the way my code handles the algorithm, since unitarity ticks slowly upward over time.

To test my code further, I tried the case of the Ma breather, which is like the Peregrine breather but with a finite period. In this case, the soliton turned into an oscillatory solution after a half a period as seen in 3.21. Further work must also be

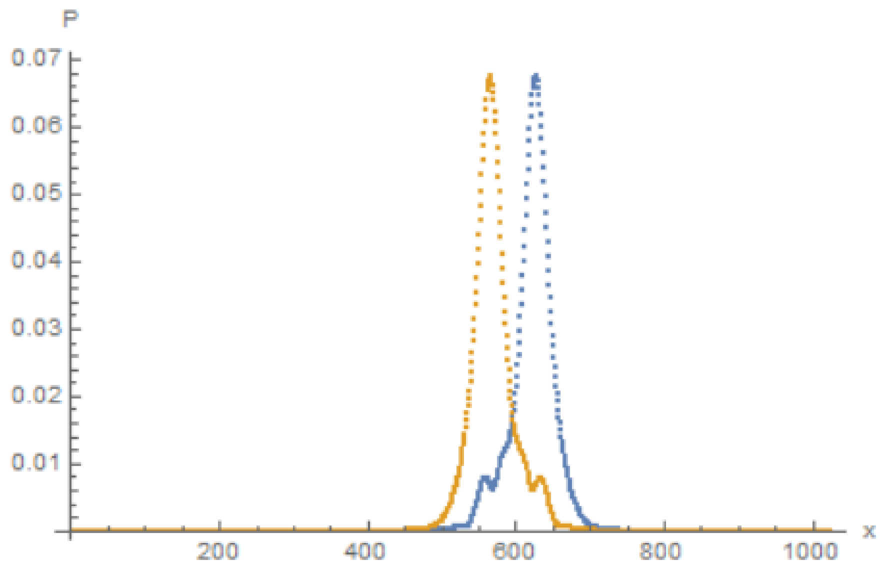

Figure 3.13: Absolute value versus position of the general solution to the Manakov system, simulated on a 1024-qubit grid at $t_1 = 500au$.

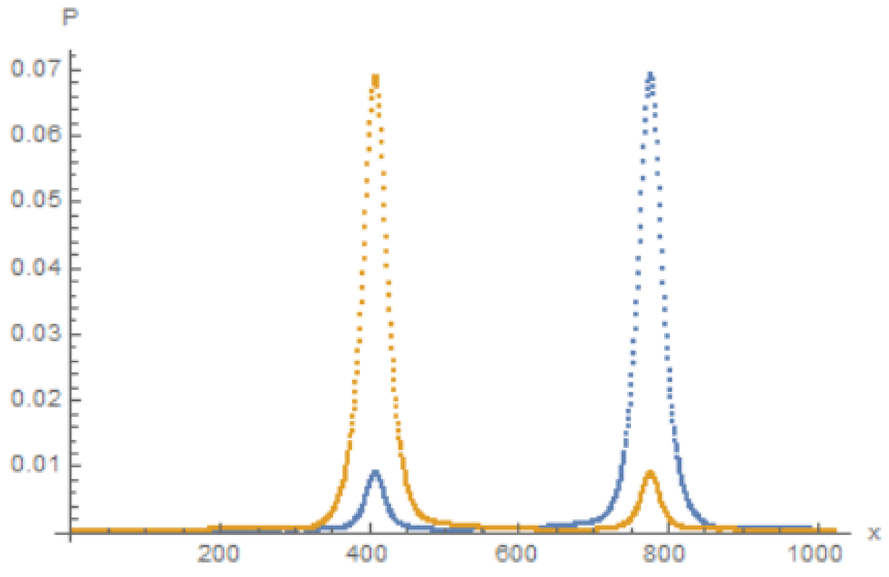

Figure 3.14: Absolute value versus position of the general solution to the Manakov system, simulated on a 1024-qubit grid at $t_2 = 1000au$.

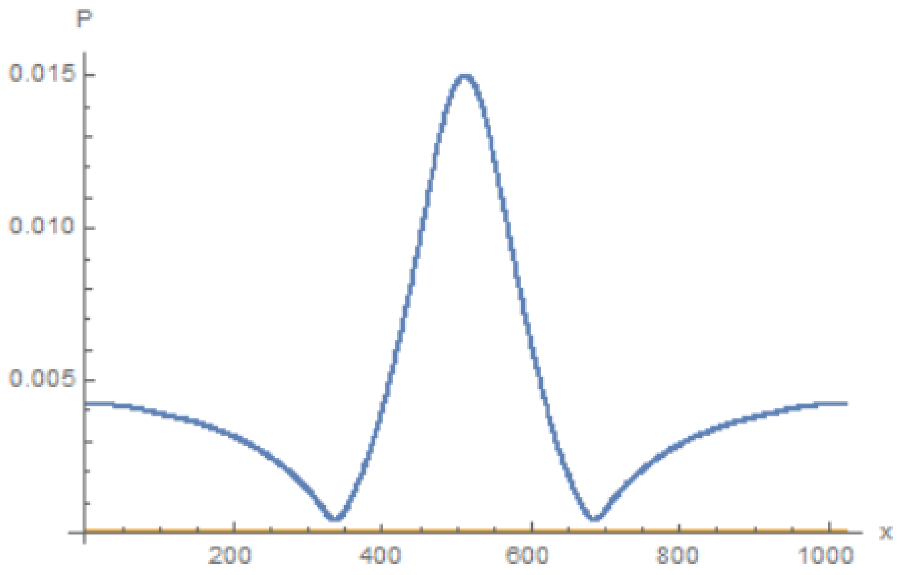

Figure 3.15: Absolute value versus position of the Peregrine solution to the uncoupled NLS, simulated on a 1024-qubit grid at $t_0 = 0au$. The solution behaves as expected at first but then dips back after vanishing to a constant solution.

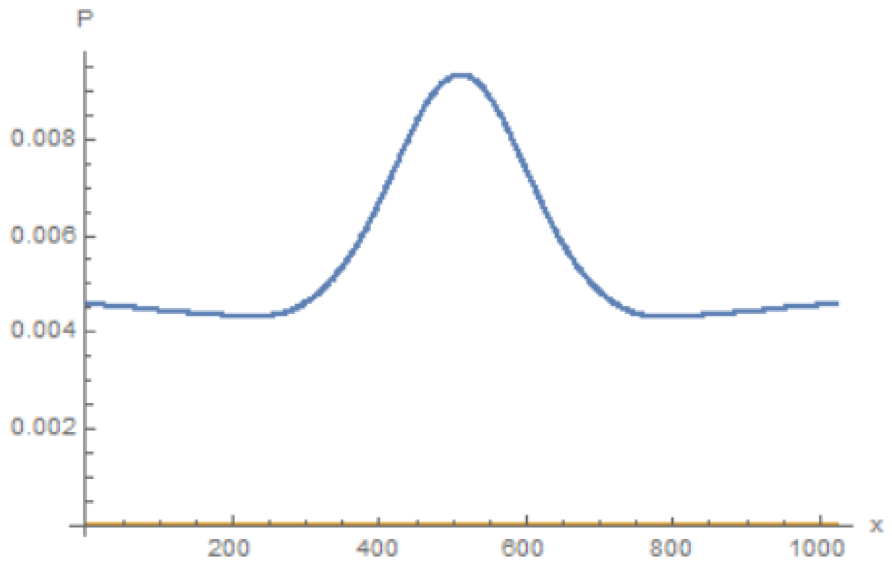

Figure 3.16: Absolute value versus position of the Peregrine solution to the uncoupled NLS, simulated on a 1024-qubit grid at $t_1 = 5000au$.

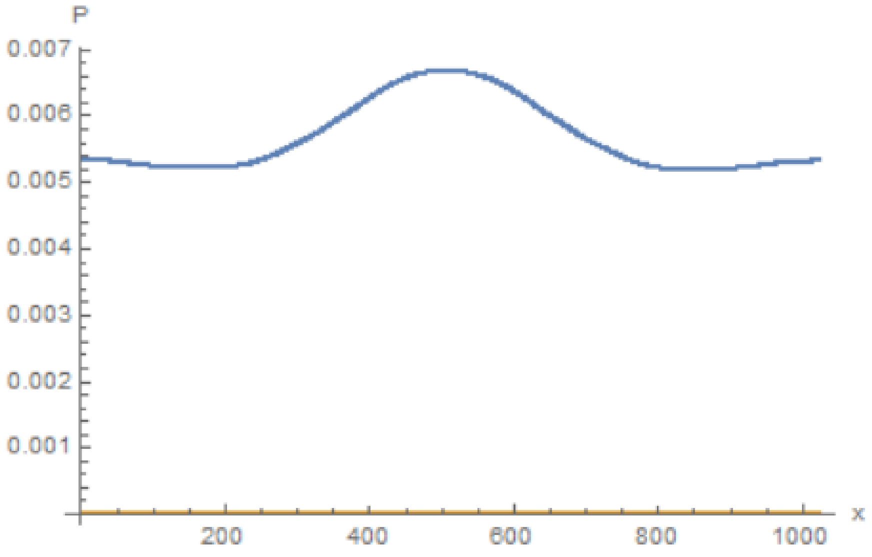

Figure 3.17: Absolute value versus position of the Peregrine solution to the uncoupled NLS, simulated on a 1024-qubit grid at $t_2 = 10000au$.

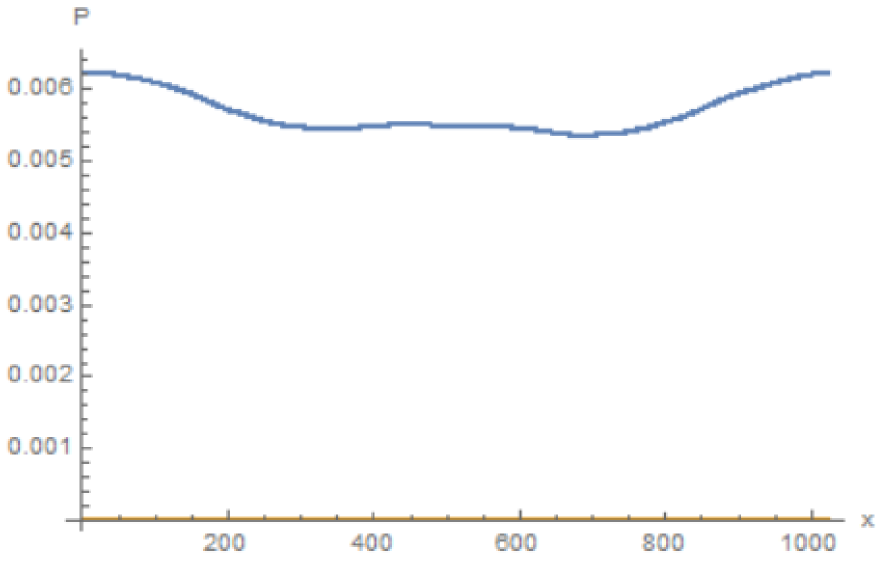

Figure 3.18: Absolute value versus position of the Peregrine solution to the uncoupled NLS, simulated on a 1024-qubit grid at $t_3 = 30000au$.

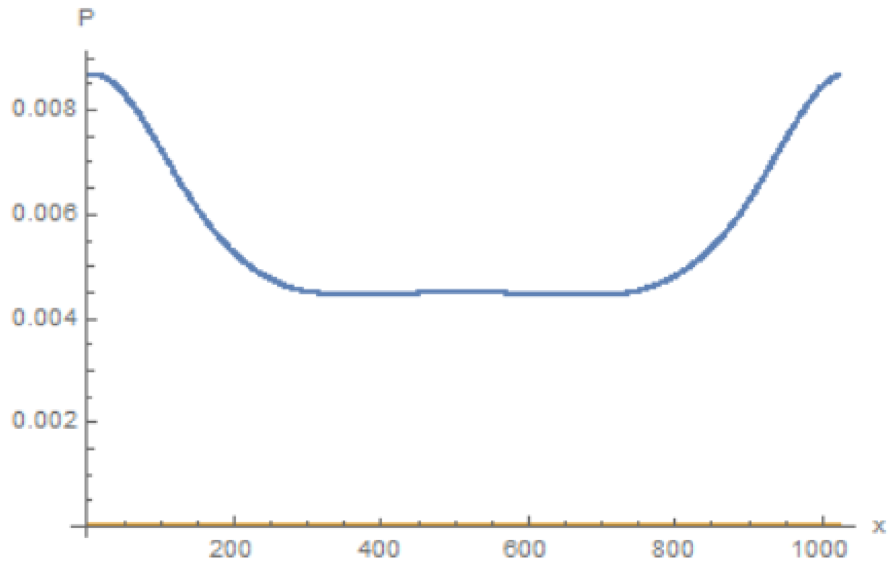

Figure 3.19: Absolute value versus position of the Peregrine solution to the uncoupled NLS, simulated on a 1024-qubit grid at $t_4 = 60000au$.

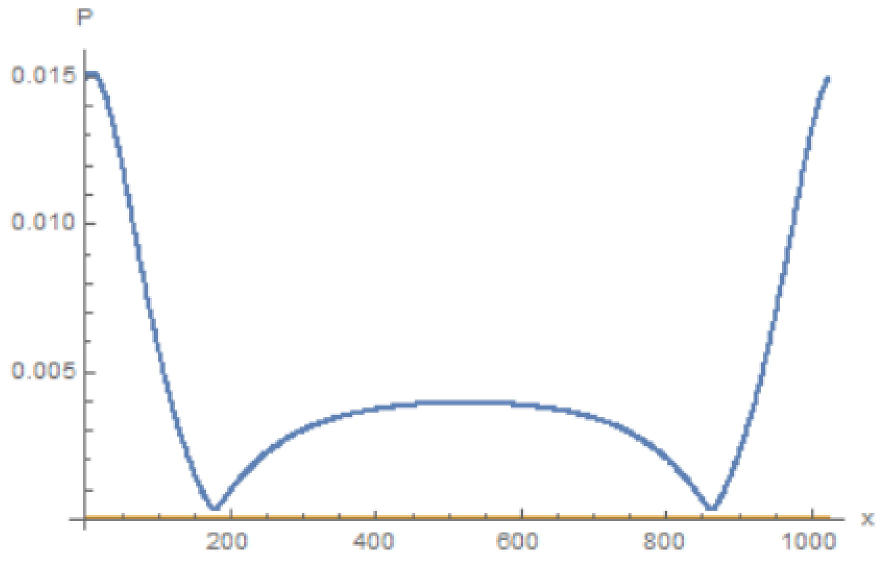

Figure 3.20: Absolute value versus position of the Peregrine solution to the uncoupled NLS, simulated on a 1024-qubit grid at $t_5 = 78000au$.

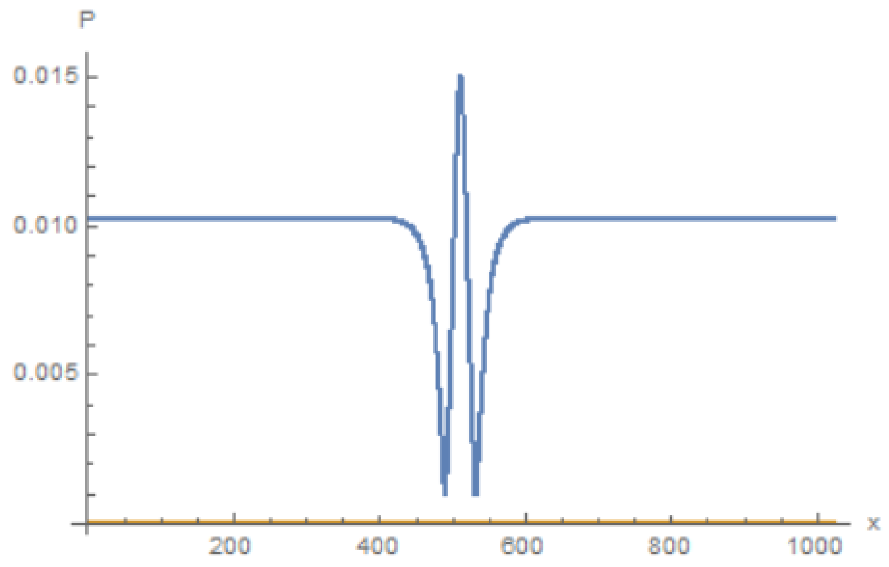

Figure 3.21: Absolute value versus position of the Peregrine solution to the uncoupled NLS, simulated on a 1024-qubit grid at $t_0 = 0au$. The solution behaves as expected at first but then exhibits more and more noise on the sides.

done to prevent the problems shown above.

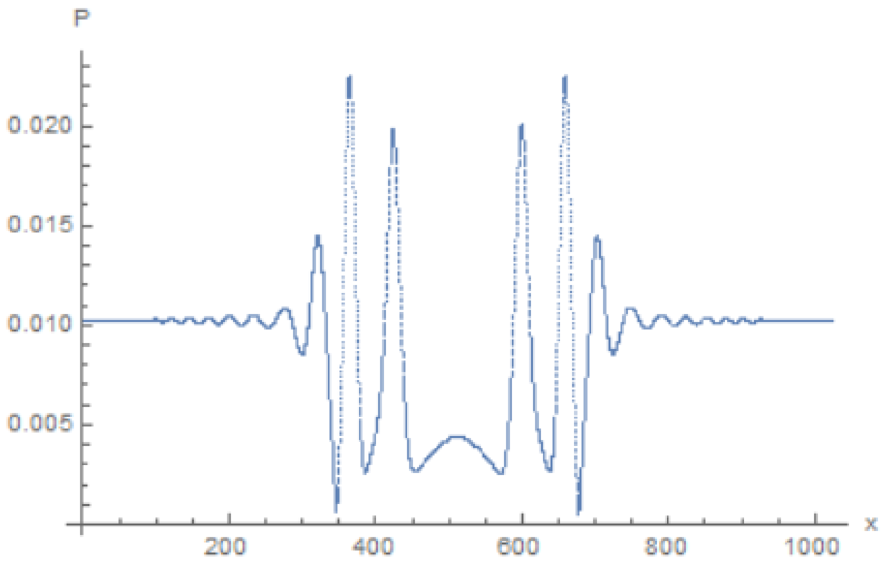

Figure 3.22: Absolute value versus position of the Peregrine solution to the uncoupled NLS, simulated on a 1024-qubit grid at $t_1 = 5000au$.

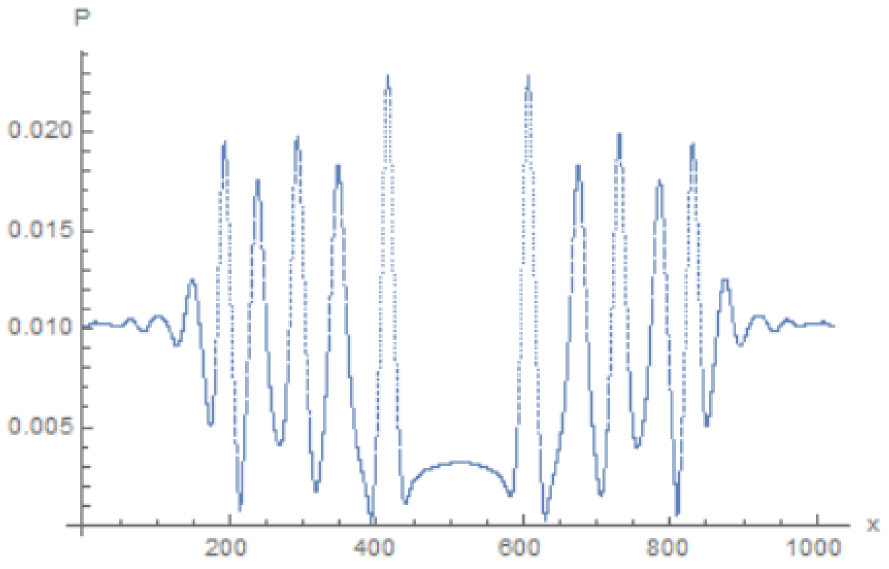

Figure 3.23: Absolute value versus position of the Peregrine solution to the uncoupled NLS, simulated on a 1024-qubit grid at $t_2 = 10000au$.

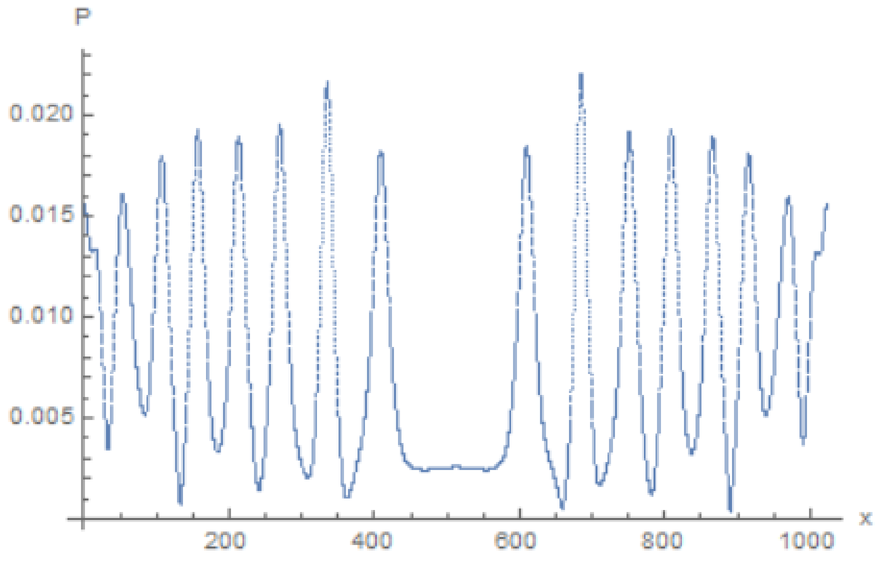

Figure 3.24: Absolute value versus position of the Peregrine solution to the uncoupled NLS, simulated on a 1024-qubit grid at $t_3 = 30000au$.

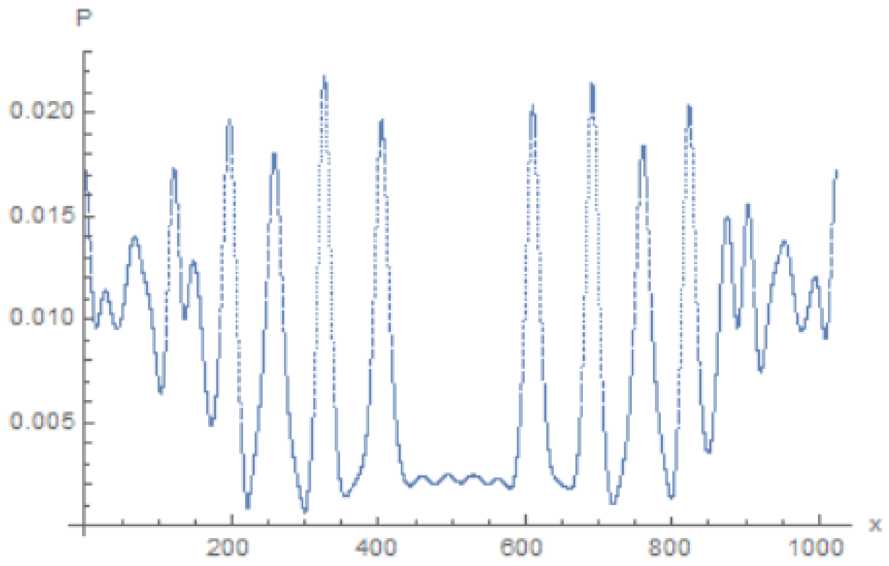

Figure 3.25: Absolute value versus position of the Peregrine solution to the uncoupled NLS, simulated on a 1024-qubit grid at $t_4 = 60000au$.

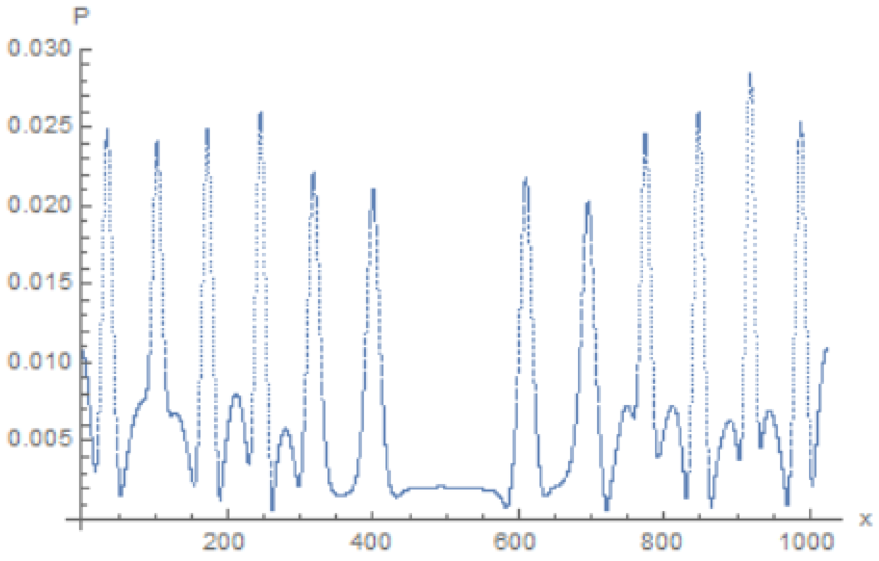

Figure 3.26: Absolute value versus position of the Peregrine solution to the uncoupled NLS, simulated on a 1024-qubit grid at $t_5 = 78000au$.

Chapter 4

Conclusions

I simulated dark soliton collisions using a qubit algorithm at various velocities. I found that at low velocity, which is less than one grid point per time iteration, the solitons exhibited the exact expected behavior. At high velocity, however, the solitons exhibit oscillations characteristic of noise in the algorithm. Some ways of fixing this include using averaging methods or avoiding velocities above what my code can handle. Meanwhile, the coupled Manakov equations showed minor soliton formation at the time of collision. This was not changed by any rescaling, suggesting that we were seeing real physics. The Peregrine and Ma breather solutions to the NLS exhibited nonphysical behavior, suggesting either that the algorithm was not exact enough or that the breathers exhibit highly nonlocal behavior that cannot be captured by the algorithm. In the future I will also test the qubit algorithm on the higher-order embedding of the Manakov equations to simulate non-perturbative solutions. A more accurate representation of the streaming operator may also overcome these problems. For example, Vahala has developed an FFT version of the qubit algorithm that simulates the NLS equation at higher amplitudes.

Appendix A

Implementing the Qubit Algorithm in Practice

I implemented the algorithm detailed in the main part of my thesis in Fortran. The code below instantiates the p and q wave functions at $t = 0$ and then applies each part of the unitary operator individually at each time step. It also prints the normalization, energy, and wave function data to different output files at a total of 100 different points in time. Note that this code was based on code written by my research group for a different project.

A.1 Code sample

The following is the Fortran code which I used to implement the qubit algorithm for several different initial conditions. The current code will instantiate both p and q solitons to simulate the pseudocollision case, although any functions can be plugged in as initial conditions.

```
! Modeling 1D solitons using a relativistic Dirac collision operator with a phase angle.
! written by Connor S.
!  ....  periodic boundary conditions, using CSHIFT (and not Armen's 'stream'-function)
module nls_manakov_nr_mod

implicit none

double precision, parameter :: pi          = acos(-1.0);                 ! Pi~
double precision, parameter :: sq2         = sqrt(2.d00);                ! (square root of 2)/2
double complex, parameter :: ii          = dcmplx(0., 1.d0);           ! complex i
integer, parameter :: GridPts     = 1024;                          ! Number of grid points
     integer, parameter            :: GridRng     = 1024;                          ! Spatial span a
```

```
double precision, parameter   :: dx          = GridRng/real(GridPts,8);    ! Grid spacing
integer, parameter :: NumRuns     = 30000                      ! number of iterations
integer, parameter :: NumOutputs  = 100                         ! number of outputs
integer, parameter :: OutputTime  = NumRuns/NumOutputs;         ! Time at which to make an outpu
double precision, parameter   :: s1Dep       = 0.0125/2.59075               ! left soliton ampli
double precision, parameter   :: s2Dep       = 0.0125/2.59075    ! right soliton amplitude
double precision, parameter   :: s1Vel       = 0.125;                ! left soliton velocity
double precision, parameter   :: s2Vel       = -0.125;                  ! right soliton velocity
double complex, parameter   :: eps         = .1;                      ! long time perturbation
double complex, parameter   :: w0          = 0.8660250;                     ! resonant frequenc
double complex, parameter   :: alpha       = .0102242;
double complex, parameter   :: phi           = 1
double complex, parameter   :: period           = 2*Sinh(2*phi)

double complex, dimension(0:GridPts) :: p, q, p0, p1, p0temp, p1temp, q0, q1, q0temp, q1temp, pc
double complex, dimension(0:GridPts) :: pcollide1122, pcollide1221, qcollide1122, qcollide1221;
! unitary collision operator
double precision, dimension(0:GridPts)  :: Vp, Vq, vpint, vqint
integer                                 :: i, it;                              ! iterators i an
character(20)                           :: file_name;                          ! file name of o
     double precision :: norm0, norm1, min0 ,abt0,qb1,norm0q,norm0p,normtp,normtq        ! norma

     contains
!  ===>
          function energy()
               double precision    :: energy;
               double complex      :: dp(0:GridPts), dq(0:GridPts);

               dp      = 0.5*(cshift(p,-1) - cshift(p,+1))/dx ! d/dx(p)
               dq      = 0.5*(cshift(q,-1)- cshift(q,+1))/dx
               energy    = dx*sum(conjg(dp)*dp+conjg(dq)*dq -((conjg(p)*p)**2+(conjg(q)*q)**2+4*
                                                   + (eps/w0)*conjg(p)*p*conjg(q)*q

               return;    ! Output the result
          end function energy
!  ---===>
 !        function current()
 !              double precision    :: pcurrent;
 !              double complex      :: dp(0:GridPts);
!
 !              dp      = (cshift(p,-1) - cshift(p,+1))/(2*dx) ! d/dx(psi1)
 !              !dq      = (cshift(q,-1) - cshift(q,+1))/(2*dx) ! d/dx(psi1)
 !              pcurrent   = dx*sum(conjg(p)*dp - p*Conjg(dp));
 !              !qcurrent   = dx*sum(conjg(q)*dq - q*Conjg(dq));
!
 !              return;  ! output the result
 !         end function current

! =====>
          ! This function will stream a 1-D qubit along a direction with the exception of the en
          ! Akin to static boundary condition as opposed to periodic d/dx at boundary = 0
          !    Numerical methods and comparison for computing dark and bright solitons in the nor
          !    Weizhu Bao, Qinglin Tang , Zhiguo Xu
     function stream(qubit, dX, BoundaryPts)
        double complex, dimension(0:GridPts)    :: stream,qubit;    ! passed qubit to the functio
integer :: dX;      ! how much along each dimension the qubit is shifted (-1, 0, +1 ..)
integer :: BoundaryPts;     ! how many points to keep unchanged at boundary

               if(BoundaryPts < 0) then ! in case the value of the boundary is negative, end exec
                    write(6,*) 'Incorrect BoundaryPts passed to stream function.';
                    stop;
               endif

               stream    = qubit;
! I will put the do loops in the if statements rather than the other way around for efficiency
               if(dX <= BoundaryPts ) then     ! proceed
                    do i = BoundaryPts - dX, GridPts - BoundaryPts - dX
                         stream(i + dX) = qubit(i);
                    enddo
               else ! if some other shift combination is sent inform user of improper input
                    write(6,*) 'Incorrect dX passed to stream function.';
                    stop;
               endif
               return;    ! Output the result
end function stream

end module nls_manakov_nr_mod


! MAIN PROGRAM BEGINS HERE
program nls_manakov_nr

     use nls_manakov_nr_mod

     ! Open files for data collection
     open(unit=1, file= "1D_nls_p_data0.txt");  ! data file housing the initial abs(psi)
open(unit=2, file= "1D_nls_norm.txt"); ! data file housing the normalization data
open(unit=3, file= "1D_nls_q_data0.txt")
     open(unit=4, file= "1D_nls_energy.txt"); ! data file housing the energy data
     open(unit=5, file= "Initial Params.txt");    ! data file housing initial parameters

    ! Output the initial conditions
!     write(5,*) 'Depth of left soliton = ',s1Dep, new_line('\n'), 'Speed of left soliton = ',s1V
! write(5,*) 'Depth of right soliton = ',s1Dep, new_line('\n'), 'Speed of right soliton = ',s2Vel
!     close(5);   ! Close file housing initial parameters

! write(6,*) 'Depth of left soliton = ',s1Dep, new_line('\n'), 'Speed of left soliton = ',s1Vel;
! write(6,*) 'Depth of right soliton = ',s2Dep, new_line('\n'), 'Speed of right soliton = ',s2Vel

! Initialize the wavefunction using the analytic solution and output it
!     do i = 0, GridPts
!          if(i <= GridPts/2) then
!              psi(i)    = (1/sq2)*(ii*s1Vel + 2*s1Dep*Tanh(s1Dep*(i - (GridPts/4))));    ! Left
!          else
!              psi(i)    = (1/sq2)*(-ii*s2Vel - 2*s2Dep*Tanh(s2Dep*(i - (3*GridPts/4))));  ! Rig
!          endif
! write(1,*) i*dx, abs(psi(i));   ! Output the position and abs(psi)
! enddo
 ! ....  using product wavefunction basis, rather than Armen's cutoff method
    do i=0, GridPts
        p(i) = exp(-ii*s2Vel*(i-300)*dx/2)*sqrt(s1Dep)/(Cosh(sqrt(s1Dep)*(i-300)*dx))
        q(i) = exp(-ii*s1Vel*(i-900)*dx/2)*sqrt(s1Dep)/(Cosh(sqrt(s1Dep)*(i-900)*dx))
       ! q(i) = 0
        !p(i) = (alpha/sqrt(3.0))*1/(Cosh(alpha*(i-800)*dx))*(1-(eps/(102*w0))*(6+(1/(Cosh(alpha
        !q(i) = (alpha/sqrt(3.0))*1/(Cosh(alpha*(i-300)*dx))*(1+(eps/(102*w0))*(6+(1/(Cosh(alpha
    enddo
close(1); ! Close the initial wavefunction file

pcp = p*conjg(p);
qcq = q*conjg(q);
Vp = -0.125*(2.*(pcp+2.*qcq)+(eps/w0)*(qcq-2.*pcp)*qcq); !Our potential term which has a (1/8) fa
    Vq = -0.125*(2.*(qcq+2.*pcp)+(eps/w0)*(2.*qcq-pcp)*pcp);
    p0      = 0.5*p;
p1     = p - p0;
q0      = 0.5*q;
q1      = q - q0;
norm0p     = dx*sum(p*conjg(p));
norm0q   = dx*sum(q*conjg(q))
     write(2,*) 0, norm0p; ! Output the initial normalization data
     write(4,*) 0, energy();

! Main time loop begins here-----------------------------------------------------------------
     do it = 1, NumRuns
          ! populate the collision operator
        pcollide1122 = cos(0.25*pi + Vp); ! the 11 and 22 components of the collision operator
pcollide1221 = -ii*sin(0.25*pi + Vp); ! the 12 and 21 components of the collision operator
qcollide1122 = cos(0.25*pi + Vq);
qcollide1221 = -ii*sin(0.25*pi + Vq);
! Begin collide and stream sequence
               ! collide the qubits (1)
            p0temp  = pcollide1122*p0 + pcollide1221*p1;
            q0temp  = qcollide1122*q0 + qcollide1221*q1;
p1      = pcollide1221*p0 + pcollide1122*p1;
q1      = qcollide1221*q0 + qcollide1122*q1;
!Stream the 0th qubit to the left
p0      = cshift(p0temp, -1);
            q0      = cshift(q0temp, -1);
            p = p0 + p1
            q = q0 + q1
          ! Update the collision operator----------
       pcp = p*conjg(p);
qcq = q*conjg(q);
       vpint = -0.125*(2.*(pcp+2.*qcq)+(eps/w0)*(qcq-2.*pcp)*qcq); !Our potential term which has
       vqint = -0.125*(2.*(qcq+2.*pcp)+(eps/w0)*(2.*qcq-pcp)*pcp);
          pcollide1122    = cos(0.25*pi + vpint);
          pcollide1221    = -ii*sin(0.25*pi + vpint);
          qcollide1122 = cos(0.25*pi + vqint);
          qcollide1221 = -ii*sin(0.25*pi + vqint);
          ! -------------------------------------
! collide the qubits (2)
            p1temp      = pcollide1221*p0 + pcollide1122*p1;
q1temp      = qcollide1221*q0 + qcollide1122*q1;
            p0  = pcollide1122*p0 + pcollide1221*p1;
            q0  = qcollide1122*q0 + qcollide1221*q1;
!Stream the 0th qubit to the left
p1      = cshift(p1temp, +1);
            q1      = cshift(q1temp, +1);
            p = p0 + p1
            q = q0 + q1
          ! Update the collision operator----------
       pcp = p*conjg(p);
qcq = q*conjg(q);
       vpint = -0.125*(2.*(pcp+2.*qcq)+(eps/w0)*(qcq-2.*pcp)*qcq); !Our potential term which has
       vqint = -0.125*(2.*(qcq+2.*pcp)+(eps/w0)*(2.*qcq-pcp)*pcp);
          pcollide1122    = cos(0.25*pi + vpint);
          pcollide1221    = -ii*sin(0.25*pi + vpint)
          qcollide1122 = cos(0.25*pi + vqint);
          qcollide1221 = -ii*sin(0.25*pi + vqint);
          ! -------------------------------------
! collide the qubits (3)
            p0temp  = pcollide1122*p0 + pcollide1221*p1;
            q0temp  = qcollide1122*q0 + qcollide1221*q1;
p1      = pcollide1221*p0 + pcollide1122*p1;
q1      = qcollide1221*q0 + qcollide1122*q1;
!Stream the 0th qubit to the left
p0      = cshift(p0temp, -1);
            q0      = cshift(q0temp, -1);
            p = p0 + p1
            q = q0 + q1
          ! Update the collision operator----------
       pcp = p*conjg(p);
qcq = q*conjg(q);
       vpint = -0.125*(2.*(pcp+2.*qcq)+(eps/w0)*(qcq-2.*pcp)*qcq); !Our potential term which has
       vqint = -0.125*(2.*(qcq+2.*pcp)+(eps/w0)*(2.*qcq-pcp)*pcp);
          pcollide1122    = cos(0.25*pi + vp);
          pcollide1221    = -ii*sin(0.25*pi + vpint)
          qcollide1122 = cos(0.25*pi + vqint);
          qcollide1221 = -ii*sin(0.25*pi + vqint);
          ! -------------------------------------
! collide the qubits (4)
            p1temp      = pcollide1221*p0 + pcollide1122*p1;
q1temp      = qcollide1221*q0 + qcollide1122*q1;
            p0  = pcollide1122*p0 + pcollide1221*p1;
            q0  = qcollide1122*q0 + qcollide1221*q1;
```

```
!Stream the 0th qubit to the left
p1      = cshift(p1temp, +1);
            q1      = cshift(q1temp, +1);
            p = p0 + p1
            q = q0 + q1
          ! Update the collision operator----------
       pcp = p*conjg(p);
qcq = q*conjg(q);
       vpint = -0.125*(2.*(pcp+2.*qcq)+(eps/w0)*(qcq-2.*pcp)*qcq); !Our potential term which has
       vqint = -0.125*(2.*(qcq+2.*pcp)+(eps/w0)*(2.*qcq-pcp)*pcp);
          pcollide1122    = cos(0.25*pi + vpint);
          pcollide1221    = -ii*sin(0.25*pi + vpint)
          qcollide1122 = cos(0.25*pi + vqint);
          qcollide1221 = -ii*sin(0.25*pi + vqint);
          ! --------------------------------------
! collide the qubits (5)
            p0temp  = pcollide1122*p0 + pcollide1221*p1;
            q0temp  = qcollide1122*q0 + qcollide1221*q1;
p1      = pcollide1221*p0 + pcollide1122*p1;
q1      = qcollide1221*q0 + qcollide1122*q1;
!Stream the 0th qubit to the left
p0      = cshift(p0temp, +1);
            q0      = cshift(q0temp, +1);
            p = p0 + p1
            q = q0 + q1
          ! Update the collision operator----------
       pcp = p*conjg(p);
qcq = q*conjg(q);
       vpint = -0.125*(2.*(pcp+2.*qcq)+(eps/w0)*(qcq-2.*pcp)*qcq); !Our potential term which has
       vqint = -0.125*(2.*(qcq+2.*pcp)+(eps/w0)*(2.*qcq-pcp)*pcp);
          pcollide1122    = cos(0.25*pi + vpint);
          pcollide1221    = -ii*sin(0.25*pi + vpint)
          qcollide1122 = cos(0.25*pi + vqint);
          qcollide1221 = -ii*sin(0.25*pi + vqint);
          ! --------------------------------------
! collide the qubits (6)
            p1temp      = pcollide1221*p0 + pcollide1122*p1;
q1temp      = qcollide1221*q0 + qcollide1122*q1;
            p0  = pcollide1122*p0 + pcollide1221*p1;
            q0  = qcollide1122*q0 + qcollide1221*q1;
!Stream the 0th qubit to the left
p1      = cshift(p1temp, -1);
            q1      = cshift(q1temp, -1);
            p = p0 + p1
            q = q0 + q1
          ! Update the collision operator----------
       pcp = p*conjg(p);
qcq = q*conjg(q);
       vpint = -0.125*(2.*(pcp+2.*qcq)+(eps/w0)*(qcq-2.*pcp)*qcq); !Our potential term which has
       vqint = -0.125*(2.*(qcq+2.*pcp)+(eps/w0)*(2.*qcq-pcp)*pcp);
          pcollide1122    = cos(0.25*pi + vpint);
          pcollide1221    = -ii*sin(0.25*pi + vpint)
          qcollide1122 = cos(0.25*pi + vqint);
          qcollide1221 = -ii*sin(0.25*pi + vqint);
          ! -------------------------------------
! collide the qubits (7)
            p0temp  = pcollide1122*p0 + pcollide1221*p1;
            q0temp  = qcollide1122*q0 + qcollide1221*q1;
p1      = pcollide1221*p0 + pcollide1122*p1;
q1      = qcollide1221*q0 + qcollide1122*q1;
!Stream the 0th qubit to the left
p0      = cshift(p0temp, +1);
            q0      = cshift(q0temp, +1);
            p = p0 + p1
            q = q0 + q1
          ! Update the collision operator----------
       pcp = p*conjg(p);
qcq = q*conjg(q);
       vpint = -0.125*(2.*(pcp+2.*qcq)+(eps/w0)*(qcq-2.*pcp)*qcq); !Our potential term which has
       vqint = -0.125*(2.*(qcq+2.*pcp)+(eps/w0)*(2.*qcq-pcp)*pcp);
          pcollide1122    = cos(0.25*pi + vpint);
          pcollide1221    = -ii*sin(0.25*pi + vpint);
          qcollide1122 = cos(0.25*pi + vqint);
          qcollide1221 = -ii*sin(0.25*pi + vqint);
          ! -------------------------------------
! collide the qubits (8)
            p1temp      = pcollide1221*p0 + pcollide1122*p1;
q1temp      = qcollide1221*q0 + qcollide1122*q1;
            p0  = pcollide1122*p0 + pcollide1221*p1;
            q0  = qcollide1122*q0 + qcollide1221*q1;
!Stream the 0th qubit to the left
p1      = cshift(p1temp, -1);
            q1      = cshift(q1temp, -1);
            p = p0 + p1
            q = q0 + q1
! end of collide/stream sequence

p  = p0 + p1;           ! update the wavefunction with the new values
q  = q0 + q1;
pcp = p*conjg(p);
qcq = q*conjg(q);
Vp = -0.125*(2.*(pcp+2.*qcq)+(eps/w0)*(qcq-2.*pcp)*qcq); !Our potential term which has a (1/8) fa
        Vq = -0.125*(2.*(qcq+2.*pcp)+(eps/w0)*(2.*qcq-pcp)*pcp);
! Data output segment
          if(mod(it, OutputTime) == 0)  then
               write(file_name, fmt = '(A11,I0,A4)') "1D_nls_q_data",it/OutputTime,".txt";  ! Ger
               open (unit=3, file=file_name); ! Open the write file
               do i = 0, GridPts
                    write(3,*) i*dx, abs(q(i));   ! write the position and wavefunction
               enddo
               write(file_name, fmt = '(A11,I0,A4)') "1D_nls_p_data",it/OutputTime,".txt";  ! Ger
               open (unit=1, file=file_name); ! Open the write file
               do i = 0, GridPts
                    write(1,*) i*dx, abs(p(i));   ! write the position and wavefunction
               enddo
               close(1); ! close the write file

 normtp =  dx*sum(p*conjg(p))
 normtq =  dx*sum(q*conjg(q))
      qbt0p = sum(p0*conjg(p0))   ;  qbt1p = sum(p1*conjg(p1))
      qbt0q = sum(q0*conjg(q0))   ;  qbt1q = sum(q1*conjg(q1))
               write(6,*) 'time = ',it, '   unitarity = ', qbt0p+qbt1p+qbt0q+qbt1q, '  normalizat
               write(2,*) it, normt ! Output the normalization data
               write(4,*) it, energy();    ! Output the energy data
          endif
     enddo
! END OF MAIN TIME LOOP------------------------------------------------------------------------
     close(2);   ! Close normalization data file
     close(4);   ! Close the energy data file
end program
```

Bibliography

[1] G. Vahala *et al* (2015), *Unitary Qubit Lattice Gas Representation of 2D and 3D Quantum Turbulence.*

[2] I.V. Barashenkov *et al*, Phys. Rev. Lett. 86, (2012), *Breathers in PT-Symmetric optical couplers.*

[3] C. Bender, S. Boettcher (2008), *Real Spectra in Non-Hermitian Hamiltonians Having PT Symmetry.*

[4] P.S. Vinayagam *et al* (2018), *New classes of solutions in the Coupled PT Symmetric Nonlocal Nonlinear Schroedinger Equations with Four Wave Mixing.*

[5] K. Dysthe, K. Trulsen (1999), *Note on Breather Solutions of the NLS as Models for Freak-Waves.*