Nonlinear Schrödinger Solver

International Centre for Theoretical Sciences

ICTS | INTERNATIONAL CENTRE *for* THEORETICAL SCIENCES

TATA INSTITUTE OF FUNDAMENTAL RESEARCH

Raghavendra Nimiwal

February 2020

Contents

Chapter 1

Overview

The N-component cubic-quintic Nonlinear Schrödinger Equation (NLS) in 1D is given by

$$i\hbar\frac{\partial\psi_k}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\psi_k}{\partial x^2} + \sum_{j=1}^{N}\alpha_{kj}|\psi_j|^2\psi_k + G_{kk}|\psi_k|^4\psi_k \tag{1.1}$$

where ψ_k is the macroscopic wave function of the $k-$th component and α is the matrix of coupling coefficients (includes both self and cross coupling) and G_{kk} is the quintic self-coupling coefficient.

The program reduces the perturbed N-component NLS system to a system of N uncoupled KdV and uses a KdV soliton initial conditions to get the initial condition for the original N-component NLS.

This is done by the following steps.

1. Madelung Transform- Converting Eq.(1.1) into its hydrodynamic form by using Madelung transform[1],

$$\psi_k(x,t) = \sqrt{\rho_k(x,t)}e^{i(m/\hbar)\int_0^x v_k(x',t)dx'} \tag{1.2}$$

We get an equation for density and velocity for each component.

$$\frac{\partial\rho_k}{\partial t} + \frac{\partial}{\partial x}(\rho_k v_k) = 0, \tag{1.3}$$

$$\frac{\partial v_k}{\partial t} = -\frac{\partial}{\partial x}\left[\frac{1}{m}\sum_{j=1}^{N}\alpha_{kj}\rho_j + \frac{v_k^2}{2} + G_{kk}\rho_k^2 - \left(\frac{\hbar^2}{2m^2}\right)\frac{\partial_x^2\sqrt{\rho_k}}{\sqrt{\rho_k}}\right]. \tag{1.4}$$

2. Linearisation- Perturbing the above hydrodynamic system about its stationary points $\rho_k = \rho_{0k} + \delta\rho_k$ and $v_k = \delta v_k$.

3. Reducing to KdV- If α matrix is positive definite the linearised dynamics after Method of Multiple Scales (Reductive Perturbation Method) can be decoupled and we get $2N$ uncoupled KdV equations (N KdV with RHS traveling solitons and N with LHS traveling solitons). Thus we can use the soliton profiles of these KdV equations to form soliton like initial condition for our original NLS system and evolve Eq.1.1 with these initial conditions. The details of Method of Multiple Scales and how to derive the KdV system are given in Chap. 3.

1.1 Prerequisite

The package contains the following files

1. *SimulationParameters.txt* : Input parameters are defined using this file.(Sec. 1.2.1)
2. *NLS_Solver.c* : The main C code which takes input from *SimulationParameters.txt* file, derives the KdV coefficients and numerically evolves the given system.
3. *global.h* : Header file for the C code which declares all the global variables.
4. *run.sh* : Bash script which links all the necessary libraries and complies the C code. It makes an executable file with the name *NLS_Solver* which is used to run the simulation.
5. *PostProcessing.ipynb* : A jupyter notebook for post-processing of the data produced by the simulation. (Sec. 1.2.2)

and the directory *Subroutines* which contains the different subroutines used in the code. The main subroutines are explained in subsequent chapters and all of them are properly commented as well. Make sure you have all files and the *Subroutines* directory in the same directory.

Before starting your first simulation, you also have to make sure that your system has the following

1. C Compiler : You can edit the *run.sh* file to use any c compiler you want. The code uses OpenMP compiler directives and library routines for parallelisation, so your compiler should also have OpenMP support. GCC by default has this support.

2. GNU Scientific Library : The C code uses GSL functions and before starting anything the user needs to download and install the GSL library and update the address where the GSL library is stored, in *run.sh* file. For this just change the variable LDir and IDir to your path.

3. Jupyter notebook with python 3 : The post-processing of the data generated by the simulation is done using jupyter notebook with python 3 kernel.

4. NumPy and SciPy Libraries : These libraries are used for most of the post-processing work.

1.2 Getting Started

The section will help you get started with the program without going into too many details of how the program is made. The details of the program will be presented in subsequent chapters.

1.2.1 Running Simulation

After installing the GSL libraries you need to update the *lib* directory path (LDir) and the *include* directory path (IDir) of your GSL in *run.sh* file. You can also change the C compiler (cc) you are using in the file and the number number of threads you want for parallelisation by changing the environment variable $OMP\_NUM\_THREADS$. Typically speed-up vs number of threads curve follow a bell curve profile because each subsequent thread increases the parallelising overheads. After updating everything the file should look like this

```
#!/bin/sh

cc=gcc
LDir=/home/raghavendra/GSL/lib
IDir=/home/raghavendra/GSL/include
CFlags="-Wall -std=c99"
Libs="-lm -lgsl -lgslcblas"

export LD_LIBRARY_PATH=$LDir
export OMP_NUM_THREADS=4
$cc $CFlags -I$IDir -c NLS_Solver.c -fopenmp
$cc -L$LDir NLS_Solver.o -o NLS_Solver $Libs -fopenmp
time ./NLS_Solver
```

To start a simulation you first need to define your system, which is done using SimulationParameters.txt file. The file contains the following fields which are used to define your system.

1. N-Coupled System : Describes the number of components in your coupled NLS system. It should be an integer value (variable type *int*).

2. M-Solitons : KdV equation has M soliton solutions, where M can be any natural number but in this program, you can choose either 1, 2 or 3 soliton solutions. It should be an integer value.
   You can also choose 0 if you wish to provide a user defined profile for the initial condition. The user defined profile has to be written in the *InitialCondition.c* subroutine. The details of which are discussed in Chap. 4.

3. Lambda : The N-component coupled NLS system reduces to N uncoupled KdV system and each uncoupled KdV has a sound speed which is determined by the eigen values of the linearized N-component NLS system. The linear operator has $2N$ distinct eigen values and they appear in pair with opposite signs. The positive eigen values corresponds to right moving chiral sector and the negative eigen values corresponds to the left moving chiral sector. The eigen values are arranged in decreasing order and Lambda determines the index for the eigenvalues ranging from 0 to $2N - 1$.

4. Domain Size : Float input which describes the computation domain or

size of the system in physical dimensions of length (**). The computational domain becomes $x \in (-\text{Domain Size}/2, \text{Domain Size}/2)$.

5. Number of gridpoints : Integer value describing the number of grid points in which the whole domain is divided. The distance between consecutive grid points is grid size Δx.

6. Final Time : Float input which describes the physical time (*with $\hbar = 1$*), which is the maximum time till you want to evolve your system.

7. delt : Float input which describes the time step used for computation. The Leap Frog explicit method imposes the stability condition that $\Delta t/\Delta x^2 \leq 1/4$ and for Runge Kutta 4th order method the stability condition is $\Delta t/\Delta x^2 \leq 0.16$. The Crank Nicolson implicit solver in unconditionally stable. If the input values of Δx and Δt does not satisfy the stability condition then the program automatically changes the value of delt. The new value is prompted on the screen when you run the program.

8. Period after which numerical results are saved : Float input for the period for saving the simulation data.

9. Epsilon : Float input which describes the small perturbation parameter. Of all the possible initial profiles, ϵ picks out the ones which balances the different terms in the perturbation expansion such that we get a KdV. Figure 1.1 shows how the initial profile changes on changing ϵ with all the other parameters remaining unchanged.

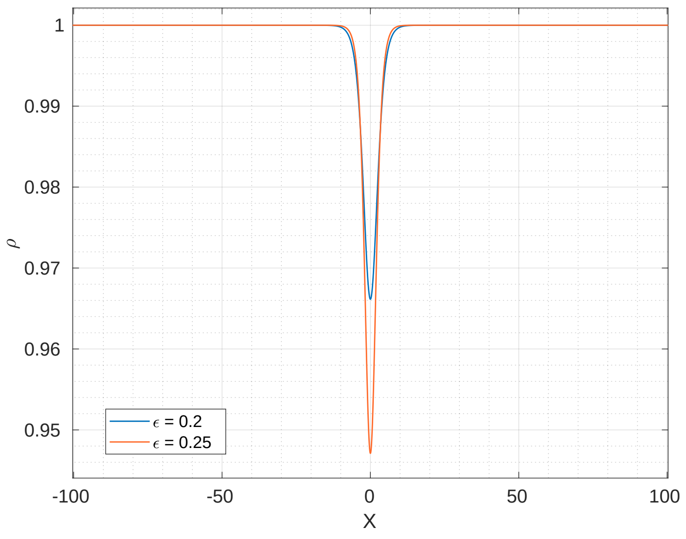


Figure 1.1: Change of initial profile with ϵ

10. Alpha_Matrix : The coupling or interaction matrix of size $N \times N$ with all distinct positive eigen values. Each element in the row should be separated by a blank space and each new row should be separated by new line. The diagonal elements should be arranged in ascending order and the off-diagonal elements should be smaller than the smallest diagonal element. This condition makes sure that the matrix is positive definite. If the matrix size is not compatible with the number of components or if the matrix is not positive definite then the program will automatically stop with an error message.

11. Rho0_Matrix : $N \times N$ diagonal matrix which represents the stationary value of $\rho_{0k} > 0$ for each component.

12. Coefficient of quintic terms: Self coupling coefficients which describes higher order intra-species interaction. The requirement is N values separated by a space.

13. KdV Soliton Parameters : A KdV soliton is characterized by two variables, determining amplitude and position. k describes the amplitude of the soliton and eta0 describes its position. For M solitons you need to put in M values of k and M values of $eta0$, where each value of k and $eta0$ are separated by a blank space. (Chap. 4).

14. Solver : The package has inbuilt 3 time stepping routines available-Leap Frog Method, Rungee Kutta 4th order Method and Crank Nicol-

son Method. The user can type LF, RK or CN respectively to tell the solver which subroutine to use for time evolution.

Once all these entries are filled, the file should look like this.

```
N-Coupled System:
3
M-solitons (0,1,2 or 3):
2
Lambda:
0
Domain size:
300.0
Number of gridpoints:
6401
Final Time:
20.0
delt:
0.0001
Period after which numerical results are saved:
0.001
Epsilon
0.2
Alpha_Matrix:
1.0 0.5 0.5
0.5 1.2 0.5
0.5 0.5 1.4
Rho0_Matrix:
1.0 0.0 0.0
0.0 0.8 0.0
0.0 0.0 1.2
Coefficients of quintic terms:
0.6 1.0 1.4
Soliton Parameters:
k=
1.4 2.0
eta0=
0.0 4.8
Time Stepping Solver (RK, LF, CN):
LF
(DISCLAIMER: PLEASE DO NOT LEAVE ANY EXTRA LINES ANYWHERE)
```

You can now execute *run.sh* to compile the program and run it.

1.2.2 Post Processing

Once the simulation is complete a new directory will be formed in the same directory where the simulation was run. The name of the new directory will be of the form *_N_S_L_tf_* where the blanks are filled by the number of components, number of solitons, the lambda value, the final time and the solver. For example *3N2S4L90tfRK* means 3 components, 2 solitons, the fifth eigenvalue in descending order, final time of simulation is 90 and the solver used for time marching is Runge Kutta 4th order.

The newly created directory will have the following files.

1. *readme.txt* : This file contains all the parameters for which the current simulation was run. It is also used as an input for the jupyter notebook *PostProcessing.ipynb*, so the user should never change the contents of this file.

2. *conservedQty1.bin* : The binary file contains the values of under area of ρ_k for each component i.e. $\int |q_k|^2 dx$ at all output time levels.

3. *conservedQty2.bin* : The binary file contains the values of total momentum of the system i.e. $\int \sum_{k=1}^{N} Im(\psi_k^* \partial_x \psi_k) dx$ at all output time levels.

4. *conservedQty3.bin* : The binary file contains the Hamiltonian of the system i.e. $\int \sum_{k=1}^{N} \left(\frac{|\partial_x \psi_k|^2}{2} + \sum_{j=1}^{N} \frac{\alpha_{kj}}{2} |\psi_k|^2 |\psi_j|^2 + \frac{G_{kk}}{2} |\psi_k|^4 |\psi_k|^2 \right) dx$ at all output time levels.

5. *_N_S_L_tfanaImag.bin* : The binary file contains the imaginary part of ψ of the derived KdV soliton at all output time levels for all components. This basically represents the linearised dynamics (after method of multiple scales) of the system.

6. *_N_S_L_tfanaReal.bin* : The binary file contains the real part of ψ of the derived KdV soliton at all output time levels for all components.

7. *_N_S_L_tfnumImag.bin* : The binary file contains the imaginary part of numerical results (KdV soliton evolved using NLS dynamics) for all components at all output time levels.

8. *_N_S_L_tfnumReal.bin* : The binary file contains the real part of numerical results (KdV soliton evolved using NLS dynamics) for all components at all output time levels.

We will use *KdV Soliton* to describe the derived KdV soliton at different time levels and *FA Soliton* to describe the soliton like numerical results we get when we evolve the initial condition using NLS dynamics.

The python notebook *PostProcessing.ipynb* reads all the above mentioned files and calculates ρ, v, tracks the soliton peaks and plots these quantities. All the variables in the notebook are described in chapter 6 so that the user can play with the output data. In its default state the notebook gives the following plots (given plots are for 2 component NLS 1 soliton system for the largest eigen value)-

Fig.1.2 shows the value of relative change in the under area of ρ for each component separately. The reference is the initial under area.

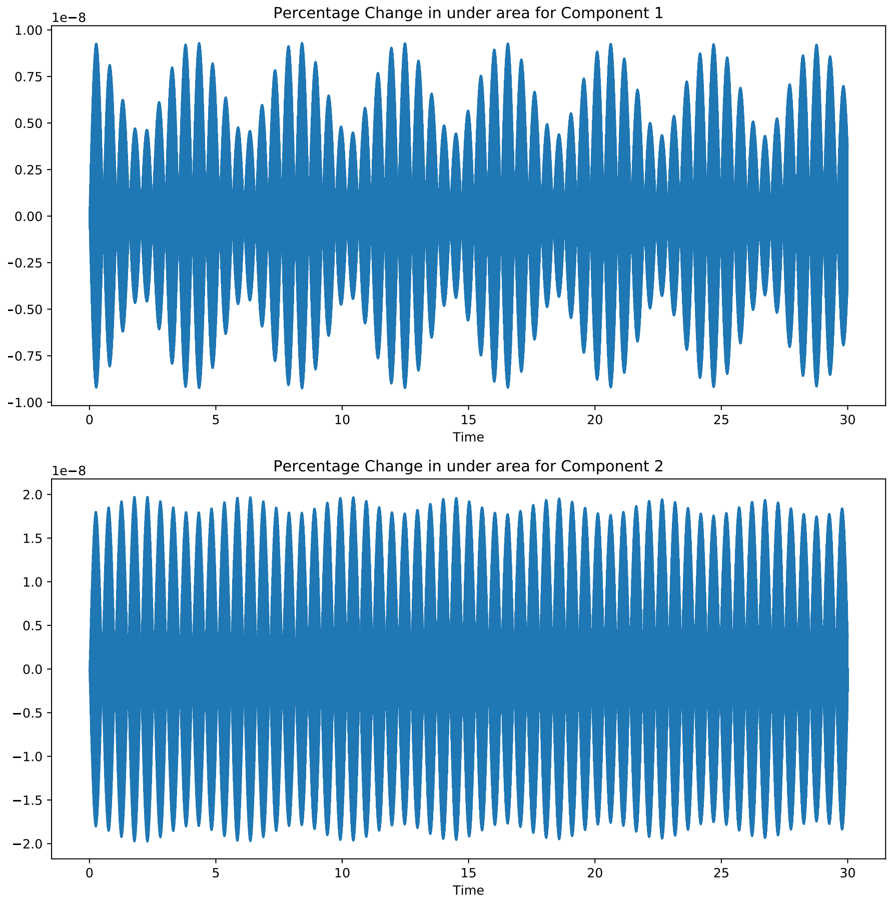


Figure 1.2: Relative change in the under area of ρ of different components

Fig.1.3 shows the value of relative change in the total momentum of the system with respect to the total initial momentum.

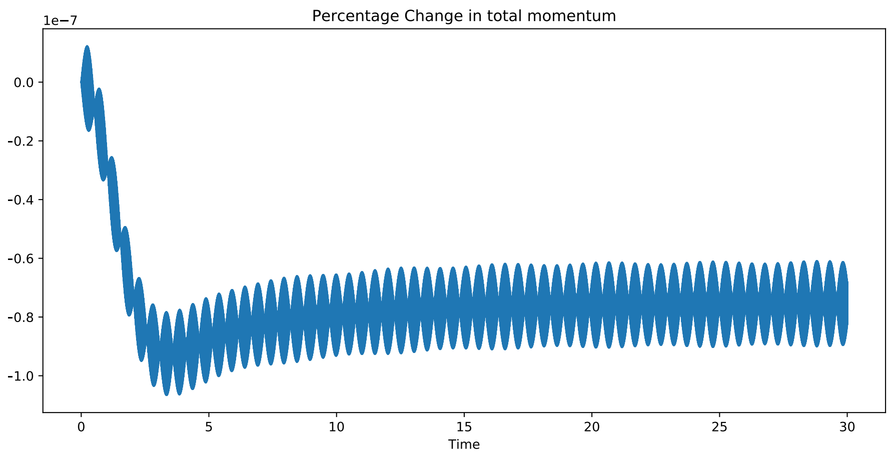


Figure 1.3: Relative change in total momentum of the system

Fig.1.4 shows the value of relative change in the Hamiltonian of the system with respect to the initial Hamiltonian.

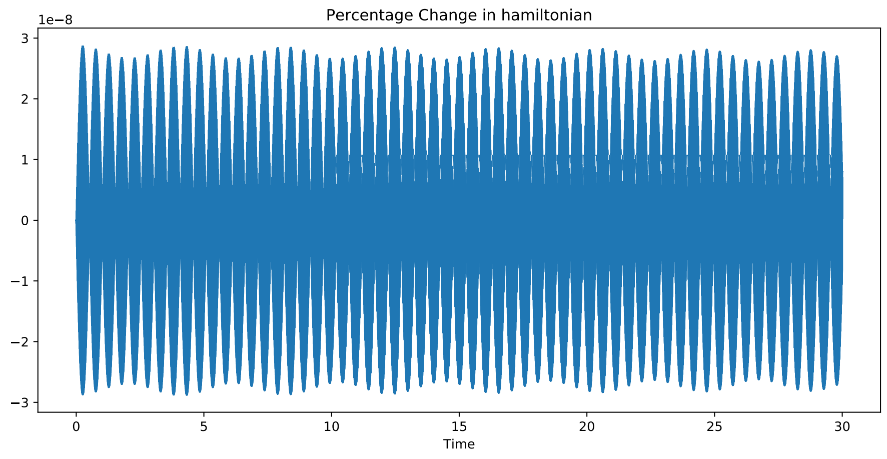


Figure 1.4: Relative change in the Hamiltonian of the system

These oscillations in the conserved quantities are due to inherent nature of leap frog time stepping scheme.

Fig.1.5 shows the ρ profiles of both components at 2 different times. Here $KdVS$ represents the KdV soliton of the derived KdV and FA represents the evolution of the initial profile we built using the derived KdV soliton.

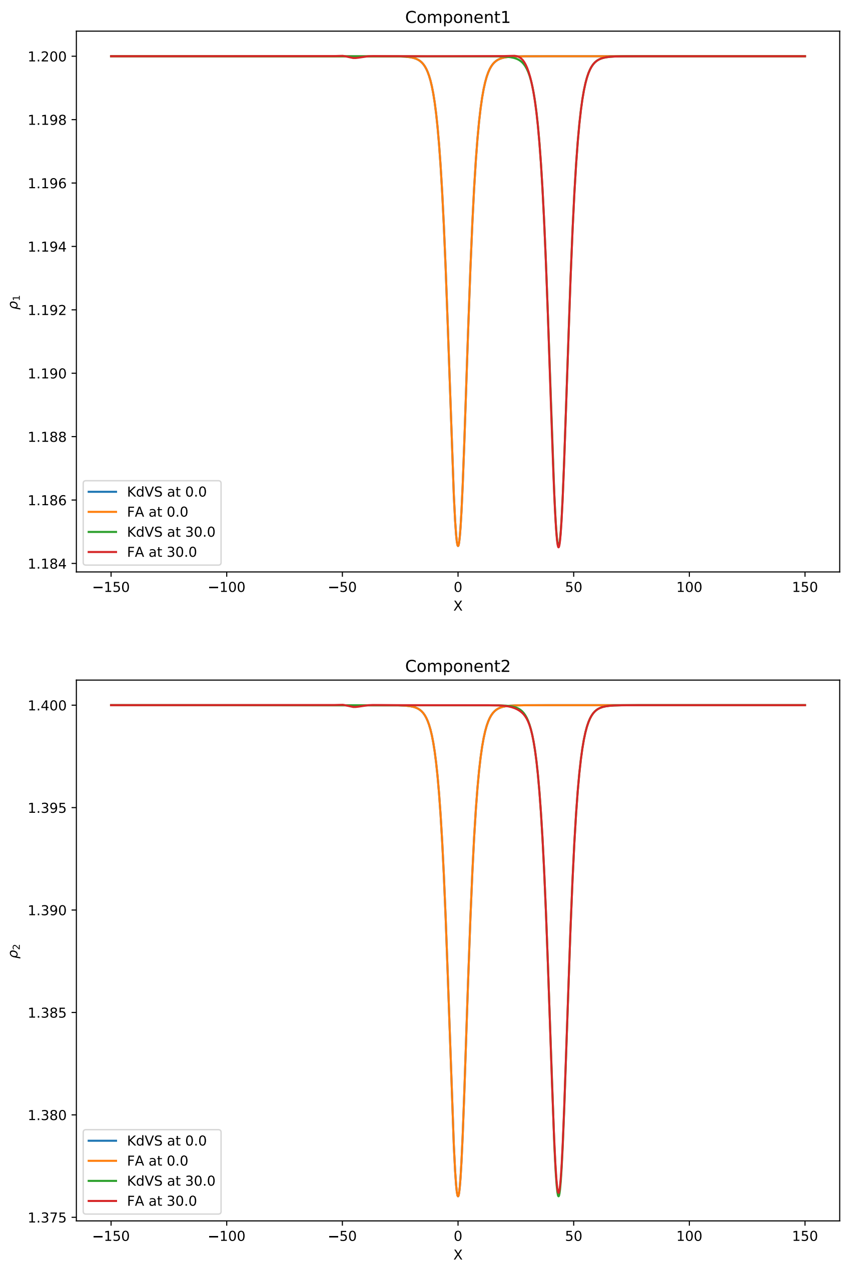


Figure 1.5: ρ profile at different times

Fig.1.6 shows the v profiles of both components at 2 different times. Here $KdVS$ represents the KdV soliton of the derived KdV and FA represents the evolution of the initial profile we built using the derived KdV soliton.

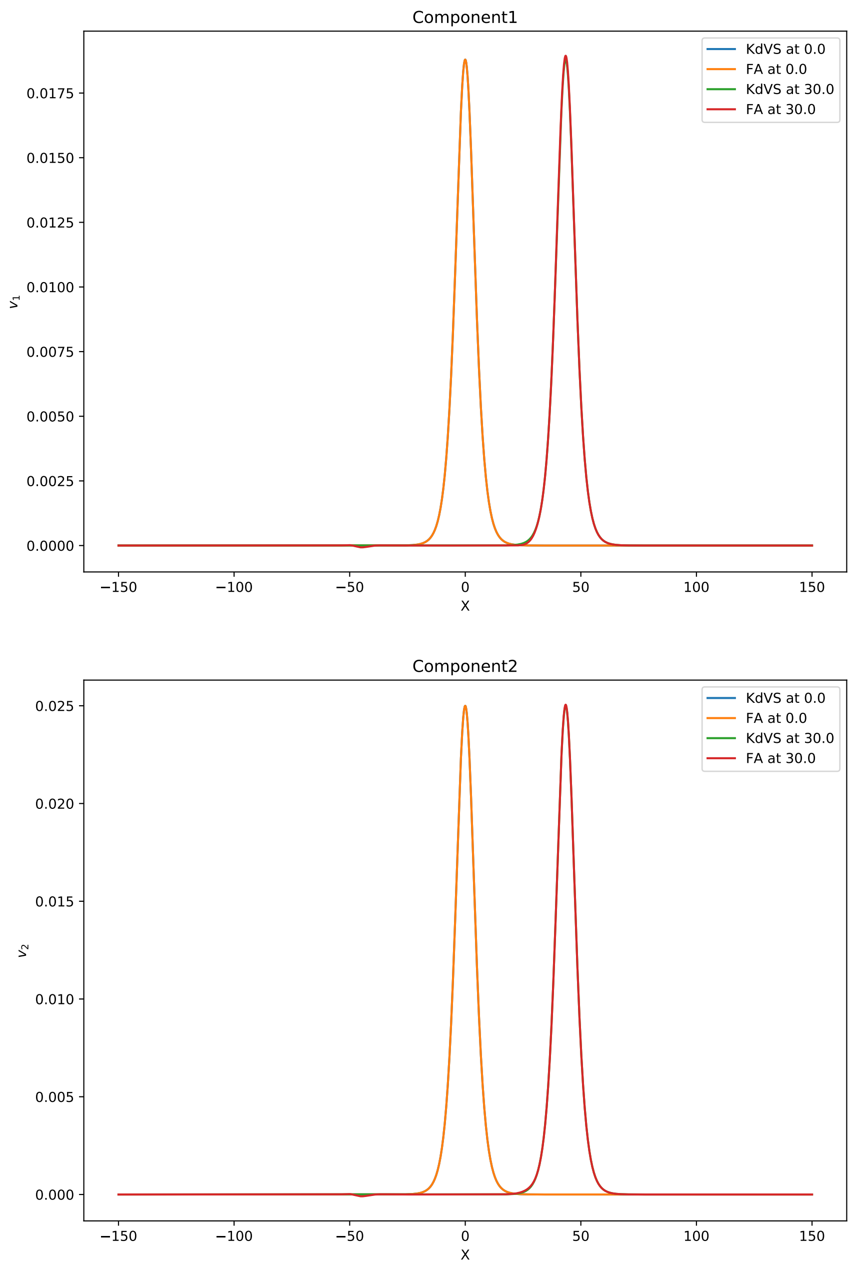
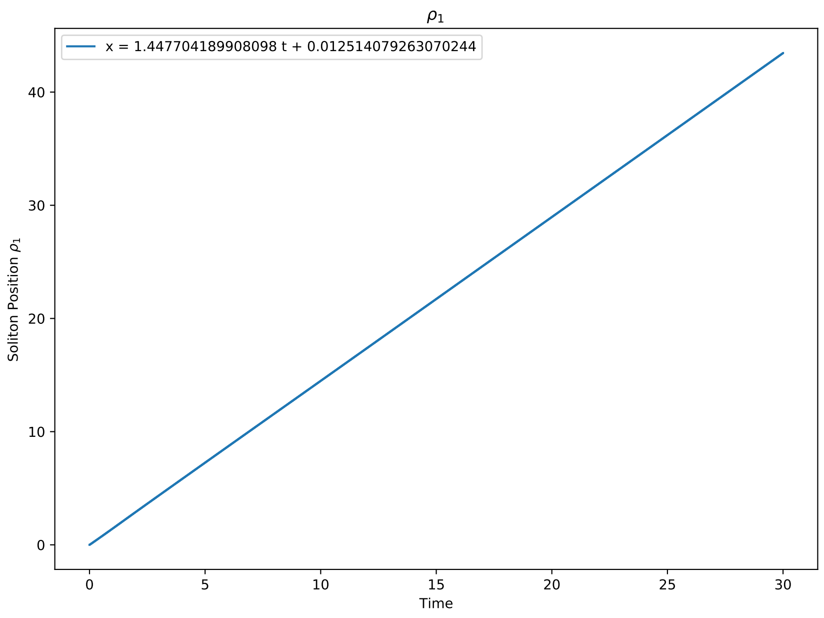


Figure 1.6: v profile at different times

As mentioned earlier that the we also track the position of the derived soliton profile as it evolves. Fig.1.7 shows the position of soliton for both the components with time. The figures show the soliton position in the ρ profile.



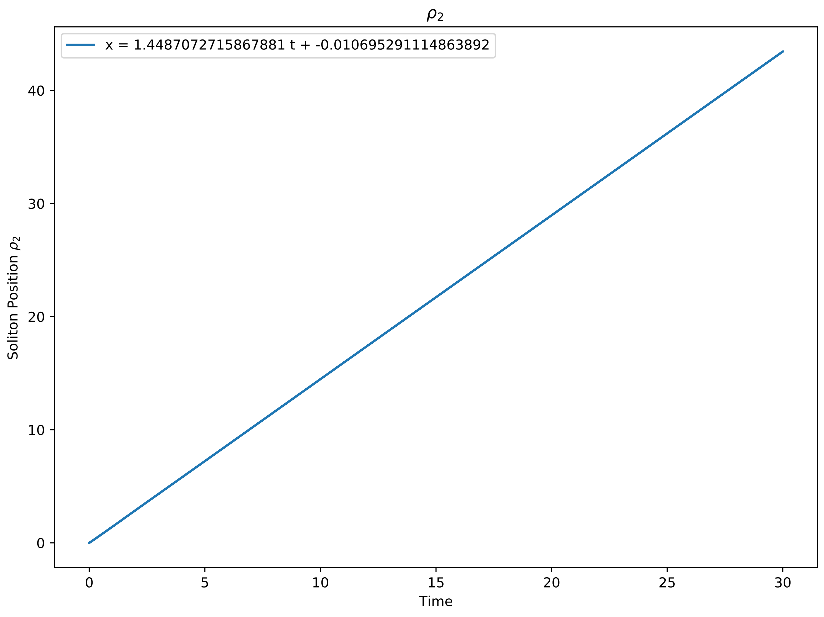


Figure 1.7: Soliton position for all components VS Time

We also track the height of the derived soliton profile as it evolves (by height we mean the absolute difference between the background value and the maximum or minimum value in the soliton profile). Fig.1.8 shows the height of the soliton in ρ profile for both components.

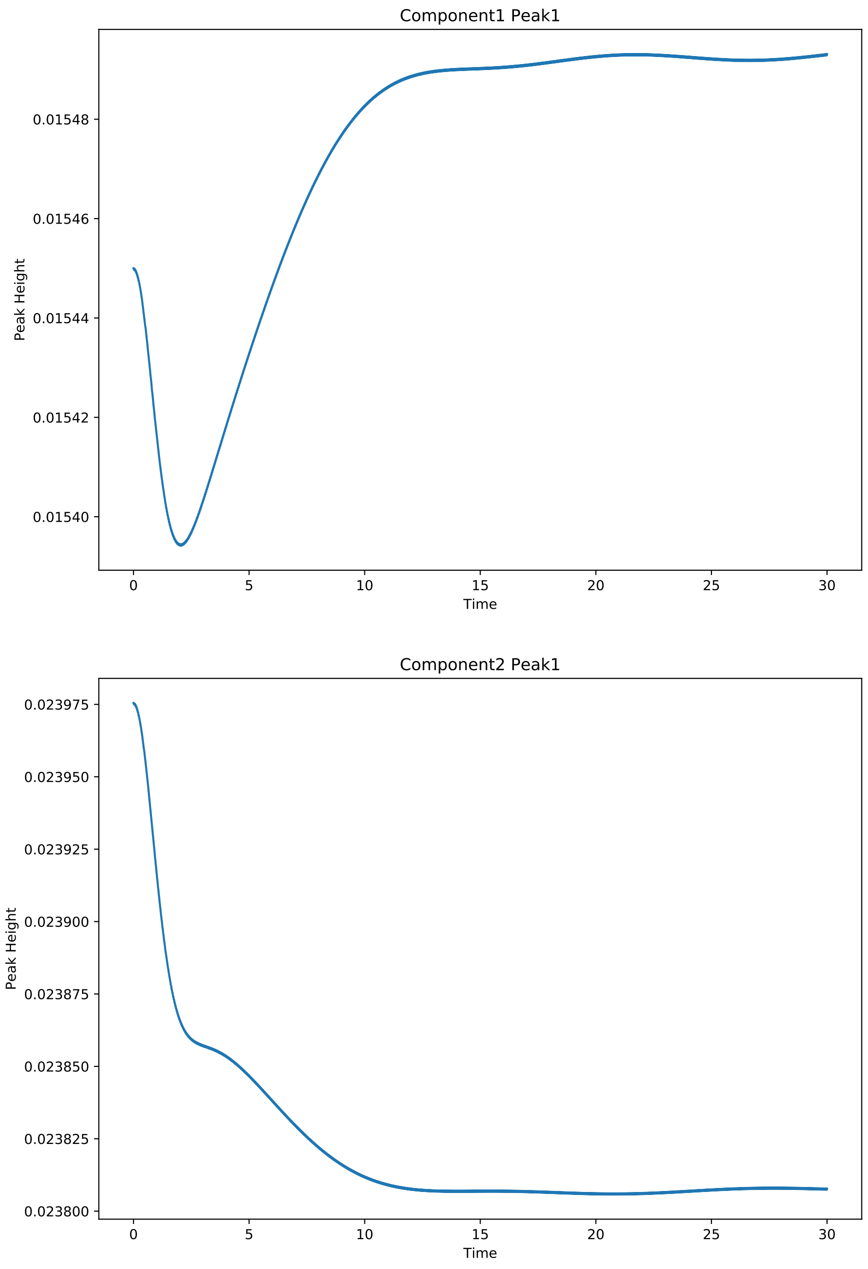


Figure 1.8: Soliton height vs Time

Chapter 2

Workflow of the package

The chapter describes the overall workflow of the solver, various global variables and functioning of various subroutines. The subroutines which require more detailed explanation have dedicated chapters later.

2.1 Global Variables

List of the global variables which define the system.

Variable Name	Description
$nCoup$	Number of components in the coupled NLS system
$mSol$	Number of KdV solitons
$lambda$	Index of the eigenvalue of linearised system for which KdV is derived
l	Length of the domain
$gridP$	Number of gridpoints
$delx$	Distance between two adjacent gridpoints
ti	Initial time of the simulation
tf	Final time of the simulation
$delt$	Time step size
$savePeriod$	Period after which the numerical results are saved
$epsilon$	Perturbation parameter
$alpha$	$nCoup \times nCoup$ array of cubic interaction coefficients
$rho0$	$nCoup \times nCoup$ array of background values of ρ_k about which the system is perturbed
$quinticCoef$	$nCoup \times nCoup$ diagonal array of quintic interactions
k	$1 \times mSol$ array of the amplitude of each KdV soliton
$eta0$	$1 \times mSol$ array of the initial position of each KdV soliton
$alphaQuintic$	The updated α matrix of the linearised system which contains contribution from quintic term as well. It is the matrix $\tilde{\alpha}$ as mentioned is 3.8
$solver$	String which stores the name of the time stepper.
X	Spatial domain

Table 2.1: Global Variables which define the system

List of global variables used to store the simulation data.

Variable Name	Description
$psiNum$	$gridP \times nCoup * (tf/savePeriod + 1)$ array which stores the numerical results after ever $savePeriod$ time starting from ti. The entries for each NLS component are stored column-wise one after the other i.e. first $tf/savePeriod + 1$ columns are for component 1, next $tf/savePeriod + 1$ for component 2 and so on.
$psiAna$	$gridP \times nCoup * (tf/savePeriod + 1)$ array which stores the actual KdV soliton of the derived KdV after ever $savePeriod$ time starting from ti The entries for each NLS component are stored column-wise one after the other i.e. first $tf/savePeriod + 1$ columns are for component 1, next $tf/savePeriod + 1$ for component 2 and so on.
$consQty1$	$nCoup \times (tf/savePeriod + 1)$ array which stores the under area of ρ after every $savePeriod$ time for all the components.
$consQty2$	$1 \times (tf/savePeriod + 1)$ array which stores the total momentum of the system after every $savePeriod$ time.
$consQty3$	$1 \times (tf/savePeriod + 1)$ array which stores the total Hamiltonian of the system after every $savePeriod$ time.

Table 2.2: Global Variables used to store the simulation data

List of global variables related to the KdV system.

Variable Name	Description
$coefNL$	Coefficient of the nonlinear term of the derived KdV.
$coefDs$	Coefficient of the dispersion term of the derived KdV.
$fScale$	Scaling variable used to convert the derived KdV to the standard 1-6-1 form.
$xiScale$	Scaling variable used to convert the derived KdV to the standard 1-6-1 form.
$tauScale$	Scaling variable used to convert the derived KdV to the standard 1-6-1 form.
$eVal$	Diagonal matrix of size $2{*}nCoup \times 2{*}nCoup$ with eigenvalues of the linearised NLS system in hydrodynamic form arranged in decreasing order. The eigenvalues determine the sound speed of the system.
$eVec$	$2{*}nCoup \times 2{*}nCoup$ matrix with eigenvectors of the linearised NLS system in hydrodynamic form. The eigenvectors are arranged column-wise corresponding to the eigenvalues in $eVal$.
$KdVSol$	Arrays of size $1 \times gridP$ which stores the KdV soliton. (arg, $theta$, f, f_0, $f1$, $f1\_0$, $f2$ are few other arrays of the same size that help in computation of the soliton profile.)

Table 2.3: Global Variables related to KdV system

Tables 2.1, 2.2 and 2.3 has an exhaustive list of all the global variables and all of these variables are stored in the file *global.h*.

2.2 Package pipeline

Fig. 2.1 shows the pipeline of the package and briefly describes each element of the pipeline.

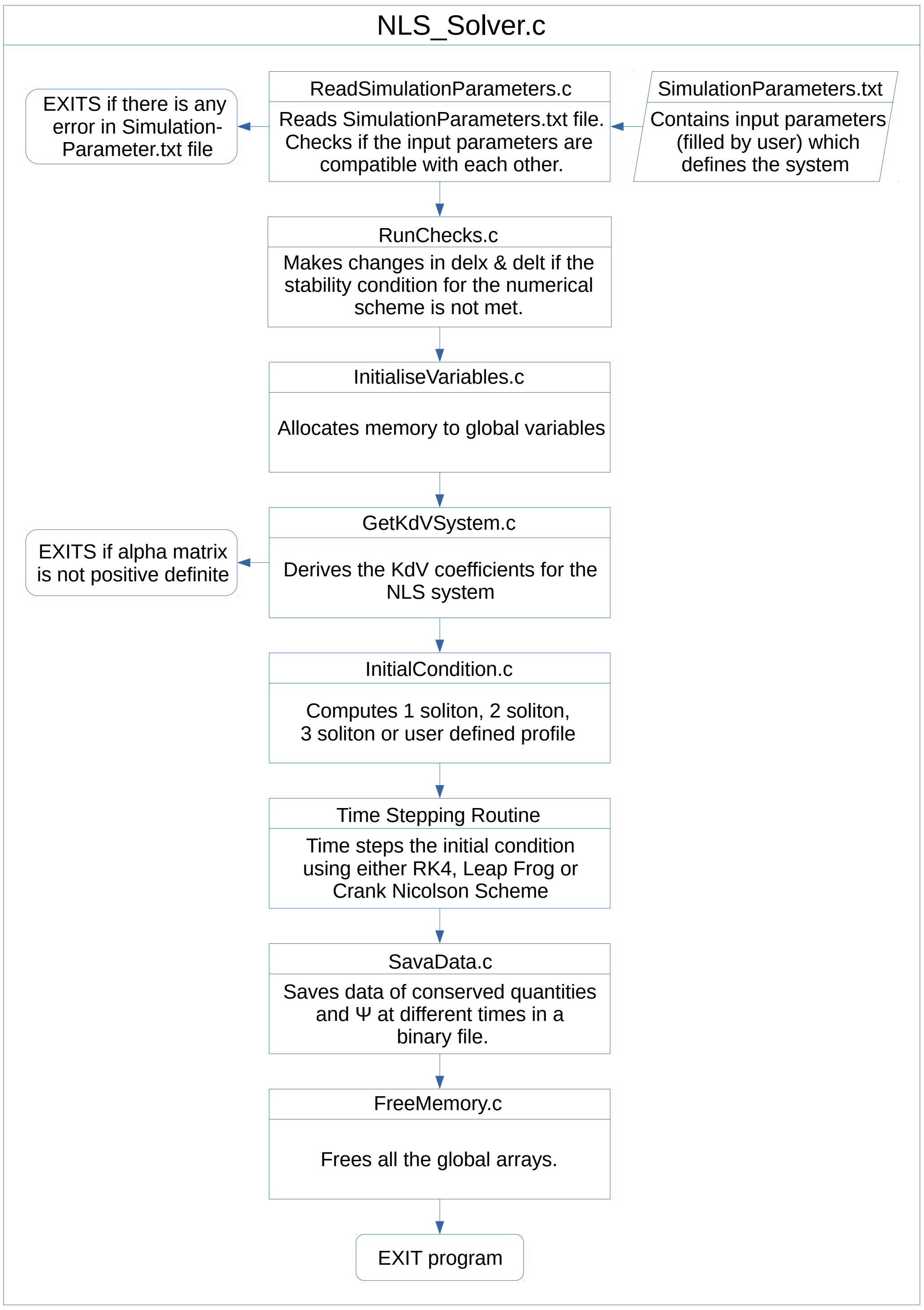

NLS_Solver.c
EXITS if there is any error in Simulation-Parameter.txt file
ReadSimulationParameters.c
Reads SimulationParameters.txt file. Checks if the input parameters are compatible with each other.
SimulationParameters.txt
Contains input parameters (filled by user) which defines the system
RunChecks.c
Makes changes in delx & delt if the stability condition for the numerical scheme is not met.
InitialiseVariables.c
Allocates memory to global variables
EXITS if alpha matrix is not positive definite
GetKdVSystem.c
Derives the KdV coefficients for the NLS system
InitialCondition.c
Computes 1 soliton, 2 soliton, 3 soliton or user defined profile
Time Stepping Routine
Time steps the initial condition using either RK4, Leap Frog or Crank Nicolson Scheme
SavaData.c
Saves data of conserved quantities and Ψ at different times in a binary file.
FreeMemory.c
Frees all the global arrays.
EXIT program

Figure 2.1: Pipeline of the package

2.2.1 ReadSimulationParameters.c

The subroutine reads *SimulationParameters.txt* file and saves the input of various fields in the global variables mentioned in table 2.1. The subroutine also runs a few checks to make sure the input data is compatible with each other. If the input data is not compatible with each other then the subroutine prompts an error message specifying where it encountered the problem and stops the program. It runs the following checks-

1. Checks if $alpha$ matrix is of size $nCoup \times nCoup$.
2. Checks if $rho0$ matrix is of size $nCoup \times nCoup$.
3. Checks if $rho0$ matrix has entries only in its diagonal and all of them are positive.
4. Checks if there are minimum $nCoup$ elements in the field *Coefficients of quintic terms*. If there are more than $nCoup$ elements then it just ignores the extra elements and reads $nCoup$ elements.
5. Checks if there are minimum $mSol$ elements in the field *Soliton Parameter k*. If there are more than $mSol$ elements then it just ignores the extra elements and reads $mSol$ elements.
6. Checks if there are minimum $mSol$ elements in the field *Soliton Parameter eta0*. If there are more than $mSol$ elements then it just ignores the extra elements and reads $mSol$ elements.

2.2.2 RunChecks.c

The subroutine checks for the stability of the explicit time steppers and makes appropriate changes if the stability condition is not met. It also makes sure that $savePeriod$ is an integral multiple of $delt$ and tf is an integral multiple of $savePeriod$ by changing the values of $delt$ and $savePeriod$ if they are not integral multiples. The subroutine runs the following checks for the stability of time steppers-

1. Leap Frog Time Stepper - For leap frog method the stability criteria is

$$\frac{\Delta t}{\Delta x^2} \leq \frac{1}{4} \tag{2.1}$$

   If this condition is not met then it uses equality in Eq. 2.1 to find Δx compatible with the original Δt and Δt compatible with the original Δx and uses both new values.

2. Rungee Kutta 4$^{\text{th}}$ order Time Stepper - For RK4 method the stability criteria is

$$\frac{\Delta t}{\Delta x^2} \leq 0.16 \tag{2.2}$$

If this condition is not met then it uses equality in Eq. 2.2 to find Δx compatible with the original Δt and Δt compatible with the original Δx and uses both new values.

Crank Nicolson is unconditionally stable so there are no checks for the scheme.

2.2.3 GetKdVsystem.c

The subroutine maps the input NLS system to an uncoupled KdV system. The details of the subroutine are given in chapter 3.

2.2.4 InitialCondition.c

The subroutine makes soliton like initial profiles for ρ and v from the derived KdV system. The details of the subroutine are given in chapters 3 and 4.

2.2.5 Time Stepping Routine

The initial soliton like profile we derived can be evolved with the following 3 time stepping routines-

1. Leap Frog Explicit Time Stepping Scheme
2. Rungee Kutta 4th Order Explicit Time Stepping Scheme
3. Crank Nicolson Implicit Time Stepping Scheme

The details of the methods, discretisation, boundary conditions are all given in details in chapter 5.

2.2.6 SaveData.c

The subroutine saves all the computational data, saved in global variables listed in table 2.2, into binary files.

Chapter 3

Getting KdV

The solver maps the input NLS system to an uncoupled KdV system. This chapter gives the details of how the KdV system is derived.

3.1 Analyticallly

We start from the N component NLS Eq.(1.1) and using Eq.(1.2), transform it to hydrodynamic form in density and velocity fields. The following details are given more clearly in [2].

$$\partial_t \rho_k + \partial_x(\rho_k v_k) = 0 \tag{3.1}$$

$$\partial_t v_k + \partial_x \sum_{j=1}^{N} \alpha_{kj}\rho_j = -\partial_x \left[\frac{v_k^2}{2} + G_{kk}\rho_k^2 - \frac{1}{4\rho_k}\frac{\partial^2 \rho_k}{\partial x^2} + \frac{1}{8\rho_k^2}\left(\frac{\partial \rho_k}{\partial x}\right)^2 \right] \tag{3.2}$$

Next the system is perturbed about its background or stationary values,

$$\rho_k = \rho_{0k} + \delta\rho_k$$

$$v_k = \delta v_k$$

The hydrodynamic equations reduces to,

$$\partial_t \delta\rho_k + \rho_{0k}\partial_x \delta v_k = -\partial_x\left(\delta\rho_k \delta v_k\right) \tag{3.3}$$

$$\begin{aligned}\partial_t \delta v_k + \partial_x \left(\sum_{j=1}^{N} \alpha_{kj} \delta\rho_j + 2G_{kk}\rho_{0k}\delta\rho_k \right) &= -\partial_x \left[\frac{\delta v_k^2}{2} + G_{kk}\delta\rho_k^2 - \frac{1}{4\rho_{0k}} \frac{\partial^2 \delta\rho_k}{\partial x^2} \right. \\ &+ \frac{\delta\rho_k}{4\rho_{0k}^2} \frac{\partial^2 \delta\rho_k}{\partial x^2} + \frac{1}{8\rho_{0k}^2} \left(\frac{\partial \delta\rho_k}{\partial x} \right)^2 \\ &- \left. \frac{\delta\rho_k}{4\rho_{0k}^3} \left(\frac{\partial \delta\rho_k}{\partial x} \right)^2 \right] \qquad (3.4)\end{aligned}$$

Let $\tilde{\alpha} = \alpha + 2G\rho$, where G and ρ are $N \times N$ diagonal matrices. Note that α and G both are interaction matrices, the diagonal elements of α matrix correspond to intra-species interaction and the off-diagonal elements correspond to inter-species interaction, whereas G the elements correspond to only intra-species interaction. The matrix elements of ρ are $\rho_{0k} > 0$. Eq.(3.3) and Eq.(3.4) can be reduces to matrix form:

$$(\partial_t + \mathcal{A}\partial_x) \begin{pmatrix} \delta\rho \\ \delta v \end{pmatrix} = -\partial_x \begin{pmatrix} \mathcal{N}_1(\delta\rho, \delta v) \\ \mathcal{N}_2(\delta\rho, \delta v) \end{pmatrix}, \qquad (3.5)$$

where

$$(\mathcal{N}_1)_k = \delta\rho_k \, \delta v_k, \qquad (3.6)$$

$$\begin{aligned}(\mathcal{N}_2)_k &= \frac{\delta v_k^2}{2} + G_{kk}\delta\rho_k^2 - \frac{1}{4\rho_{0k}} \frac{\partial^2 \delta\rho_k}{\partial x^2} \\ &+ \frac{\delta\rho_k}{4\rho_{0k}^2} \frac{\partial^2 \delta\rho_k}{\partial x^2} + \frac{1}{8\rho_{0k}^2} \left(\frac{\partial \delta\rho_k}{\partial x} \right)^2 - \frac{\delta\rho_k}{4\rho_{0k}^3} \left(\frac{\partial \delta\rho_k}{\partial x} \right)^2 \qquad (3.7)\end{aligned}$$

$$\mathcal{A} = \begin{pmatrix} \mathbf{0}_{N\times N} & \rho \\ \tilde{\alpha} & \mathbf{0}_{N\times N} \end{pmatrix} \qquad (3.8)$$

The coupled NLS maps to uncoupled KdV in the long wavelength regime where the non linearity and dispersion effects (the first three terms of the R.H.S of Eq.(3.7)) are balanced. To get this balance, we need to re-scale the space time variables,

$$\partial_t \to \epsilon\partial_t \quad \partial_x \to \epsilon\partial_x \quad \delta\rho \to \epsilon^2\delta\rho \quad \delta v \to \epsilon^2\delta v$$

where ϵ is a dimensionless small parameter. This scaling is equivalent to assuming the following for the original physical variables

$$\rho_k = \rho_k^{(0)} + \epsilon^2\delta\rho_k(\epsilon x, \epsilon t) \qquad (3.9)$$

$$v_k = \epsilon^2\delta v_k(\epsilon x, \epsilon t) \qquad (3.10)$$

This leads to

$$(\partial_T + \mathcal{A}\partial_X)\begin{pmatrix}\delta\rho\\ \delta v\end{pmatrix} = -\epsilon^2\partial_X\begin{pmatrix}\mathcal{N}_1(\delta\rho,\delta v)\\ \mathcal{N}_2(\delta\rho,\delta v,\epsilon^2)\end{pmatrix}, \tag{3.11}$$

where $X = \epsilon x, T = \epsilon t$ and

$$\begin{aligned}\mathcal{N}_2(\delta\rho,\delta v,\epsilon^2) = {}& \frac{\delta v_k^2}{2} + G_{kk}\delta\rho_k^2 - \frac{1}{4\rho_{0k}}\frac{\partial^2\delta\rho_k}{\partial x^2}\\ &+\epsilon^2\frac{\delta\rho_k}{4\rho_{0k}^2}\frac{\partial^2\delta\rho_k}{\partial x^2} + \epsilon^2\frac{1}{8\rho_{0k}^2}\left(\frac{\partial\delta\rho_k}{\partial x}\right)^2 - \epsilon^4\frac{\delta\rho_k}{4\rho_{0k}^3}\left(\frac{\partial\delta\rho_k}{\partial x}\right)^2\end{aligned} \tag{3.12}$$

To recall, due to the above scaling the first three terms of the R.H.S of $\mathcal{N}_2$ falls in the same order of ϵ, these terms will contribute to the non linearity and dispersion effects. Thus this scaling might lead us to KdV equation.

We now solve the above equation perturbatively. Assuming an expansion in ϵ^2 for the solution,

$$\begin{pmatrix}\delta\rho\\ \delta v\end{pmatrix} = \begin{pmatrix}\delta\rho^{(0)}\\ \delta v^{(0)}\end{pmatrix} + \epsilon^2\begin{pmatrix}\delta\rho^{(1)}\\ \delta v^{(1)}\end{pmatrix} \tag{3.13}$$

Substituting into the Eq.(3.11) and collecting the terms to lowest order, we get,

$$(\partial_T + \mathcal{A}\partial_X)\begin{pmatrix}\delta\rho^{(0)}\\ \delta v^{(0)}\end{pmatrix} = \begin{pmatrix}0\\ 0\end{pmatrix}. \tag{3.14}$$

If $\tilde{\alpha}$ matrix is positive definite then $\mathcal{A}$ will have real and distinct eigen values and they come in pair with opposite signs [2]. This also means that $\mathcal{A}$ is diagonalisable and hence we can decouple the system. The positive eigen values correspond to the right chiral sector and the negative eigen values correspond to the left chiral sector.

Since $\mathcal{A} = V\tilde{\Lambda}V^{-1}$ is diagonalisable, $\tilde{\Lambda}$ is an $2N \times 2N$ diagonal matrix with eigen values arranged in decreasing order such that, from 0 to $N-1$ are the positive eigenvalues and from N to $2N-1$ are the negative eigenvalues. V is the matrix with its columns as the eigen vectors of $\mathcal{A}$ matrix. Then Eq.(3.14) is equivalent to (decoupled)

$$\left(\partial_T + \tilde{\Lambda}\partial_X\right)V^{-1}\begin{pmatrix}\delta\rho^{(0)}\\ \delta v^{(0)}\end{pmatrix} = \begin{pmatrix}0\\ 0\end{pmatrix}, \tag{3.15}$$

which has a solution

$$V^{-1}\begin{pmatrix}\delta\rho^{(0)}\\ \delta v^{(0)}\end{pmatrix} = f_j^{(0)}(X - \tilde{\lambda}_j T)e_j, \tag{3.16}$$

or

$$\begin{pmatrix}\delta\rho^{(0)}\\ \delta v^{(0)}\end{pmatrix} = f_j^{(0)}(\xi)Ve_j, \tag{3.17}$$

where $\xi = X - \tilde{\lambda}_j T$ and $f_j^{(0)}(\xi)$ is an arbitrary function, $\tilde{\lambda}_j$ is any of the eigenvalues of $\mathcal{A}$ and e_j is the canonical unit vector in $\mathbb{R}^{2N}$ corresponding to the eigen value. This shows that any function of the form $f_j^{(0)}(X - \tilde{\lambda}_j T)$ will satisfy the equation at the ϵ^0 order but this solution does not have any higher order effects.

Now we want to add some constraints on this solution such that it satisfies the equation with higher order effects as well. To do this we will assume $f_j^{(0)}$ depends on $X - \tilde{\lambda}_j T$ as well as a new slow time scale $\tau = \epsilon^2 T = \epsilon^3 t$. Introducing this new slow time will force the time (τ) derivative of the function to appear in the next order equation. Hence

$$\begin{pmatrix}\delta\rho^{(0)}\\ \delta v^{(0)}\end{pmatrix} = f_j^{(0)}(X - \tilde{\lambda}_j T, \tau)Ve_j. \tag{3.18}$$

The equations at order ϵ^2 will now become

$$\begin{aligned}(\partial_T + \mathcal{A}\partial_X)\begin{pmatrix}\delta\rho^{(1)}\\ \delta v^{(1)}\end{pmatrix} &= -\partial_\tau \begin{pmatrix}\delta\rho^{(0)}\\ \delta v^{(0)}\end{pmatrix}\\ &\quad - \partial_X \begin{pmatrix}\mathcal{N}_1(\delta\rho^{(0)}, \delta v^{(0)})\\ \mathcal{N}_2(\delta\rho^{(0)}, \delta v^{(0)}, 0)\end{pmatrix},\end{aligned} \tag{3.19}$$

where

$$(\mathcal{N}_1)_k = \delta\rho_k^{(0)}\delta v_k^{(0)} \tag{3.20}$$

$$(\mathcal{N}_2)_k = \frac{(\delta v_k^{(0)})^2}{2} + G_{kk}(\delta\rho_k^{(0)})^2 - \frac{1}{4\rho_{0k}}\frac{\partial^2 \delta\rho_k^{(0)}}{\partial x^2} \tag{3.21}$$

which is equivalent to

$$\begin{aligned}\left(\partial_T + \tilde{\Lambda}\partial_X\right)V^{-1}\begin{pmatrix}\delta\rho^{(1)}\\ \delta v^{(1)}\end{pmatrix} &= -\partial_\tau V^{-1}\begin{pmatrix}\delta\rho^{(0)}\\ \delta v^{(0)}\end{pmatrix}\\ &\quad - \partial_X V^{-1}\begin{pmatrix}\mathcal{N}_1(\delta\rho^{(0)}, \delta v^{(0)})\\ \mathcal{N}_2(\delta\rho^{(0)}, \delta v^{(0)}, 0)\end{pmatrix}.\end{aligned} \tag{3.22}$$

Notice this is a linear inhomogeneous equation for the order ϵ^2 correction to $\delta\rho, \delta v$. The L.H.S of above equation is a non-invertible matrix. In order to get a solution for a non-invertible system the R.H.S should be orthogonal to the null space of the adjoint of the linear operator $\left(\partial_T + \tilde{\Lambda}\partial_X\right)V^{-1}$. This condition is called the Fredholm alternative. Moreover, the adjoint of the linear operator on the left-hand side has a null space: precisely those functions of the form $\psi(X - \tilde{\lambda}_j T)e_j$.

The R.H.S is essentially a known function since every term on the right hand side can be written in terms of $f_j^{(0)}(X - \tilde{\lambda}_j T, \tau)$. Hence we have the solvability condition

$$\left\langle e_j, -\partial_\tau V^{-1}\begin{pmatrix}\delta\rho^{(0)}\\ \delta v^{(0)}\end{pmatrix} - \partial_X V^{-1}\begin{pmatrix}\mathcal{N}_1(\delta\rho^{(0)}, \delta v^{(0)})\\ \mathcal{N}_2(\delta\rho^{(0)}, \delta v^{(0)}, 0)\end{pmatrix}\right\rangle = 0,$$

where e_j being the $j-$th unit vector of the $2N \times 2N$ identity matrix. The above equation can be simplified using the expression for the zeroth order solution Eq.(3.18),

$$\partial_\tau f_j^{(0)} + \left\langle e_j, \partial_X V^{-1}\begin{pmatrix}\mathcal{N}_1(\delta\rho^{(0)}, \delta v^{(0)})\\ \mathcal{N}_2(\delta\rho^{(0)}, \delta v^{(0)}, 0)\end{pmatrix}\right\rangle = 0. \tag{3.23}$$

The inner product of e_j and V^{-1} in Eq. 3.23 will give a row vector (j^{th} row of V^{-1}) which when multiplied by the row vector containing the nonlinear and dispersion terms ($\mathcal{N}$) will give a scalar (the nonlinear and the dispersion terms). Rewriting $\mathcal{N}_1, \mathcal{N}_2$ entirely in terms of $f_j^{(0)}$ we get the required KdV equation,

$$\partial_\tau f_j^{(0)} + K_{NL} f_j^{(0)} \partial_\xi f_j^{(0)} + K_{Ds} \partial_\xi^3 f_j^{(0)} = 0 \tag{3.24}$$

The above equation is obtained using Eq.(3.18), Eq.(3.20) and Eq.(3.21). This is true for each j and hence we have derived a system of uncoupled KdV equations, where K_{NL} and K_{Ds} are the coefficients for the non-linear and dispersion terms respectively.

To summarize, the perturbation in ρ and v, i.e. $\delta\rho$ and δv to the lowest order in ϵ are given by

$$\begin{pmatrix}\delta\rho^{(0)}\\ \delta v^{(0)}\end{pmatrix} = f_j^{(0)}(\xi, \tau) V e_j,$$

where $f_j^{(0)}(\xi, \tau)$ satisfies

$$\partial_\tau f_j^{(0)} + K_{NL} f_j^{(0)} \partial_X f_j^{(0)} + K_{Ds} \partial_X^3 f_j^{(0)} = 0$$

Therefore density and velocity vectors from Eq. 3.9 and 3.10 are given by

$$\begin{pmatrix}\rho\\ v\end{pmatrix} = \begin{pmatrix}\rho_0\\ 0\end{pmatrix} + \epsilon^2 f_j^{(0)}(\epsilon x - \tilde{\lambda}_j \epsilon t, \epsilon^3 t) V e_j + \mathcal{O}(\epsilon^4), \tag{3.25}$$

where λ_j is eigenvalue associated with the eigenvector Ve_j (e_j being the $j-$th unit vector of the $2N \times 2N$ identity matrix and V being the matix of all the eigenvectors).

3.2 GetKdVSystem subroutine

This section shows the details of how the system of uncoupled KdV is derived in the solver. Also note that the variables used in this section like $nCoup$, $rho0$, etc are the actual names of the variables used in the code. As seen in the previous section we first linearise the given nonlinear system. The linearised hydrodynamic equation in matrix form can be written as-

$$\partial_T \delta\rho + RR\partial_X \delta\rho + RV\partial_X \delta v = 0$$
$$\partial_T \delta v + VR\partial_X \delta\rho + VV\partial_X \delta v = 0$$

The subroutine has these 4 matrices- RR, RV, VR and VV of size $nCoup \times nCoup$. In our case RR & VV are null matrices, RV is diagonal matrix $rho0$ and VR is the matrix $alphaQuintic$ as seen in Eq. 3.8.

It is important to note that if the solver is used for some other RHS in Eq. 1.1 then these RR, RV, VR and VV matrices need to be changed accordingly.

We then fined the eigenvalues and eigenvectors of the block matrix

$$\begin{pmatrix} RR_{nCoup\times nCoup} & RV_{nCoup\times nCoup} \\ VR_{nCoup\times nCoup} & VV_{nCoup\times nCoup} \end{pmatrix}$$

If any of the eigenvalues come out to be imaginary the solver prompts an error message and stops. For each eigenvalue (sound speed) the NLS maps to a KdV equation. Thus an N-component NLS maps to $2N$ KdV equations.

Now to get the KdV equations we need to implement Eq.3.23. To implement this we have two arrays $N1$ and $N2$ of sizes $nCoup\times 2$. The first column of $N1$ contains coefficient of nonlinear term in $\delta\rho^{(1)}$ equation i.e. nonlinear terms of Eq.3.20 and second column contains the coefficient of dispersion term in $\delta\rho^{(1)}$ equation i.e. dispersion terms of Eq.3.20, which are 0 in this case. Similarly the first column of $N2$ contains coefficient of nonlinear term in $\delta v^{(1)}$ equation i.e. nonlinear terms of Eq.3.21 and second column contains the coefficient of dispersion term in $\delta v^{(1)}$ i.e. the dispersion term in Eq. 3.21 equation.

Therefore we can fill the entries of $N1$ and $N2$ using Eq 3.18, 3.20 & 3.21. It is to be noted that if the user changes the RHS of Eq. 1.1 then the $N1$ and $N2$ should be filled accordingly.

$N1$ and $N2$ actually are vectors, here they have two columns so that we can calculate $coefNL$ and $coefDs$ separately. Consider a block matrix

$$N = \begin{pmatrix} N1 \\ N2 \end{pmatrix}$$

Now to get $coefNL$ for j^{th} eigenvalue we take the inner product of j^{th} row of V^{-1} and first column of N and multiply the whole thing by 2 (we get an extra 2 because $\partial_X(f_j^{(0)})^2 = 2f_j^{(0)}\partial_X f_j^{(0)}$). To get $coefDs$ for j^{th} eigenvalue we take the inner product of j^{th} row of V^{-1} and second column of N. This gives us the KdV equation.

$$\partial_\tau f_j^{(0)} + K_{NL} f_j^{(0)} \partial_X f_j^{(0)} + K_{Ds} \partial_X^3 f_j^{(0)} = 0$$

which is same as

$$\partial_\tau f_j^{(0)} + K_{NL} f_j^{(0)} \partial_\xi f_j^{(0)} + K_{Ds} \partial_\xi^3 f_j^{(0)} = 0 \tag{3.26}$$

since $\partial X/\partial\xi = 1$. K_{NL} is stored as $coefNL$ and K_{Ds} is stored as $coefDs$ in the code.

Now we scale Eq.3.26 to convert it into the standard 1-6-1 form of KdV. To do this we scale the variables as

$$\tau = C_\tau \tilde{\tau} \tag{3.27}$$

$$\xi = C_\xi \tilde{\xi} \tag{3.28}$$

C_τ is stored as $tauScale$ and C_ξ is stored as $xiScale$ in the code.

With these scaling Eq. 3.26 becomes

$$\partial_{\tilde{\tau}} f_j^{(0)} + \frac{C_\tau}{C_\xi} K_{NL} f_j^{(0)} \partial_{\tilde{\xi}} f_j^{(0)} + \frac{C_\tau}{C_\xi^3} K_{Ds} \partial_{\tilde{\xi}}^3 f_j^{(0)} = 0 \tag{3.29}$$

Therefore for the KdV to be in 1-6-1 form

$$C_\xi = \sqrt{\frac{6K_{Ds}}{K_{NL}}} \tag{3.30}$$

$$C_\tau = \frac{6}{K_{NL}} \sqrt{\frac{6K_{Ds}}{K_{NL}}} \tag{3.31}$$

The subroutine *getKdVSystem.c* does the above computation to get $coefNL$ and $coefDs$ and also gives values to the scaling variables $xiScale$ and $tauScale$.

Chapter 4

Initial Conditions

As shown in the earlier section

$$\begin{aligned}\rho_k &= \rho_{0k} + \epsilon^2 V_{k,j} f(\xi,\tau)\\ v_k &= \epsilon^2 V_{k+N,j} f(\xi,\tau)\end{aligned} \tag{4.1}$$

where

$$\begin{aligned}\rho_k &= \text{Density of the } k^{th} \text{ condensate} & \xi &= \epsilon x - \tilde{\lambda}_j \epsilon t\\ v_k &= \text{Velocity of the } k^{th} \text{ condensate} & \tau &= \epsilon^3 t\\ V_{m,n} &= m^{th} \text{ component of } n^{th} \text{ eigen vector} & N &= \text{N-coupled system}\\ f(\xi,\tau) &= \text{Solution to the derived KdV equation} & &\end{aligned}$$

The KdV we get is

$$\partial_\tau f_j^{(0)} + K_{NL} f_j^{(0)} \partial_\xi f_j^{(0)} + K_{Ds} \partial_\xi^3 f_j^{(0)} = 0$$

which after scaling to 1-6-1 form becomes

$$\partial_{\tilde{\tau}} f_j^{(0)} + \frac{C_\tau}{C_\xi} K_{NL} f_j^{(0)} \partial_{\tilde{\xi}} f_j^{(0)} + \frac{C_\tau}{C_\xi^3} K_{Ds} \partial_{\tilde{\xi}}^3 f_j^{(0)} = 0$$

where

$$\begin{aligned}C_\xi &= \sqrt{\frac{6K_{Ds}}{K_{NL}}}\\ C_\tau &= \frac{6}{K_{NL}} \sqrt{\frac{6K_{Ds}}{K_{NL}}}\end{aligned}$$

Also recall $\psi_k = \sqrt{\rho_k} e^{i \int_0^x v_k dx'}$. We want to find the soliton solution for $f_j^{(0)}$ and construct the ψ from it.

Lets first see the soliton solutions for a KdV of the form

$$\partial_{\tilde{\tau}} f_j^{(0)} + 6 f_j^{(0)} \partial_{\tilde{\xi}} f_j^{(0)} + \partial_{\tilde{\xi}}^3 f_j^{(0)} = 0 \tag{4.2}$$

there exists soliton solutions which are given below

1. 1 Soliton solution

$$f_j^{(0)}(\tilde{\xi}, \tilde{\tau}) = 2\partial_{\tilde{\xi}}^2 (log\, u)$$
$$u = 1 + e^{\eta_1}$$
$$\eta_1 = k_1 \tilde{\xi} - k_1^3 \tilde{\tau} - \eta_1^{(0)}$$

2. 2 Soliton solution

$$f_j^{(0)}(\tilde{\xi}, \tilde{\tau}) = 2\partial_{\tilde{\xi}}^2 (log\, u)$$
$$u = 1 + e^{\eta_1} + e^{\eta_2} + e^{\eta_1 + \eta_2 + A_{12}}$$
$$\eta_i = k_i \tilde{\xi} - k_i^3 \tilde{\tau} - \eta_i^{(0)}$$
$$e^{A_{ij}} = \left(\frac{k_i - k_j}{k_i + k_j} \right)^2$$

3. 3 Soliton solution

$$f_j^{(0)}(\tilde{\xi}, \tilde{\tau}) = 2\partial_{\tilde{\xi}}^2 (log\, u)$$

$$u = 1 + e^{\eta_1} + e^{\eta_2} + e^{\eta_3} + e^{\eta_1+\eta_2+A_{12}} + e^{\eta_2+\eta_3+A_{23}} + e^{\eta_3+\eta_1+A_{31}} + e^{\eta_1+\eta_2+\eta_3+A_{12}+A_{23}+A_{31}}$$

$$\eta_1 = k_i \tilde{\xi} - k_i^3 \tilde{\tau} - \eta_i^{(0)}$$
$$e^{A_{ij}} = \left(\frac{k_i - k_j}{k_i + k_j} \right)^2$$

Here k_i determines the height of the soliton and η_i determines the initial position of the soliton.

4.1 Initial condition for ψ from one soliton solution

The one soliton solution for our case will be

$$\begin{aligned} f_j^{(0)}(\tilde{\xi},\tilde{\tau}) &= 2\partial_{\tilde{\xi}}^2(log\, u) \\ &= 2\left[\frac{1}{u}\frac{\partial^2 u}{\partial\tilde{\xi}^2} - \frac{1}{u^2}\left(\frac{\partial u}{\partial\tilde{\xi}}\right)^2\right] \end{aligned} \tag{4.3}$$

where

$$u = 1 + e^{\eta_1}$$

and

$$\eta_1 = k_1\tilde{\xi} - k_1^3\tilde{\tau} - \eta_1^{(0)}$$

Simplifying the expressions we get

$$\frac{\partial u}{\partial\tilde{\xi}} = k_1 e^{k_1\tilde{\xi}-k_1^3\tilde{\tau}-\eta_1^{(0)}}$$

and

$$\frac{\partial^2 u}{\partial\tilde{\xi}^2} = k_1^2 e^{k_1\tilde{\xi}-k_1^3\tilde{\tau}-\eta_1^{(0)}}$$

We get $f_j^{(0)}(x,t)$ by changing all the variables to the original variables x and t. Hence we get

$$\rho_k(x,t) = \rho_{0k} + \epsilon^2 f_j^{(0)}(x,t)V_{k,j} \tag{4.4}$$

$$v_k(x,t) = \epsilon^2 f_j^{(0)}(x,t)V_{N+k,j} \tag{4.5}$$

which means the bump or the dip in the profiles of ρ_k and v_k are determined by $V_{k,j}$ and $V_{N+k,j}$

Now to find the phase of ψ_k, (lets call it ϕ_k), we need to integrate v_k.

$$\begin{aligned} \phi_k &= i\int_0^x v_k(x',t)dx' \\ &= i\epsilon^2 V_{N+k,j}\int_0^x f_j^{(0)}(x',t)dx' \end{aligned} \tag{4.6}$$

We know that

$$\begin{aligned} f_j^{(0)} &= 2\frac{\partial}{\partial\tilde{\xi}}\left(\frac{1}{u}\frac{\partial u}{\partial\tilde{\xi}}\right) \\ f_j^{(0)} &= 2\frac{\partial}{\partial x}\left(\frac{1}{u}\frac{\partial u}{\partial\tilde{\xi}}\right)\frac{C_\xi}{\epsilon} \end{aligned}$$

$$\begin{aligned}\phi_k &= i\epsilon^2 V_{N+k,j}\int_0^x f_j^{(0)}dx' \\ &= i\epsilon^2 V_{N+k,j}\frac{2C_\xi}{\epsilon}\left[\frac{1}{u}\frac{\partial u}{\partial\tilde{\xi}}\right]_0^x \end{aligned} \tag{4.7}$$

And hence we have computed ψ for one soliton solution.

The same calculation is done to get 2 soliton and 3 soliton initial condition.

One important thing to remember here is that we can get KdV coefficients K_{NL} and K_{Ds} such that C_ξ and C_τ are purely imaginary numbers but $\eta_i = k_i\xi/C_\xi - k_i^3\tau/C_\tau + \eta_i^{(0)}$ should be real. To make sure that η_i is real, when C_ξ and C_τ are imaginary, the subroutine *getKdVsystem.c* changes the value of k_i to purely imaginary by multiplying it by $\sqrt{-1}$.

Note that the derived initial condition is a function of the variable η_i which after simplification becomes

$$\eta_i = x\left[\frac{k_i\epsilon}{C_\xi}\right] - t\left[\frac{k_i\epsilon\tilde{\lambda_j}}{C_\xi} + \frac{k_i^3\epsilon^3}{C_\tau}\right] - \eta_i^0$$

which gives us the speed of the soliton

$$\begin{aligned} v_j &= \tilde{\lambda_j} + \frac{C_\xi}{C_\tau}k_i^2\epsilon^2 \\ &= \tilde{\lambda_j} + \frac{K_{NL}}{6}k_i^2\epsilon^2 \end{aligned}$$

Recall that this soliton solution is the solution of the linearised system after applying Method of Multiple Scales but we evolve these initial conditions with our original equation (Eq.1.1) therefore the initial condition when evolved with Eq.1.1 and the derived solitons are expected to get out of sync in the long time regime.

4.2 InitialCondition.h subroutine

The subroutine computes the value of ψ_k at any given time t from the derived KdV or any other user defined profile. Since $\psi_k = \sqrt{\rho_k}e^{i\int_0^x v_k dx'}$, you can use any function for v_k and ρ_k to describe the initial profile.

The program gives the following 4 options to set the initial profile.

1. $mSol = 1$: The solver uses 1 soliton solution of the derived KdV to form ψ_k.

2. $mSol = 2$: The solver uses 2 soliton solution of the derived KdV to form ψ_k.

3. $mSol = 3$: The solver uses 3 soliton solution of the derived KdV to form ψ_k.

4. $mSol = 0$: By choosing $mSol = 0$ the subroutine lets the user define profiles for ρ_k and v_k

The solver does the same calculation described in the section above. Since *GetKdVSystem* subroutine calculates and saves eigenvalues, eigenvectors and *xiScale* & *tauScale* in global variables, *InitialCondition.c* subroutine uses these values to compute the KdV soliton solution and saves it in the variable *KdVSol*. It then computes ψ_k from *KdVSol*.

Let's see some example initial conditions- Consider 2 coupled NLS system for which we are trying to generate initial conditions which are similar to 2 solitons.

```
N-Coupled System:
2
M-solitons (0,1,2 or 3):
2
Epsilon
0.2
Alpha_Matrix:
1.0 0.5
0.5 1.2
Rho0_Matrix:
1.2 0.0
0.0 1.5
Coefficients of quintic terms:
0.0 0.0
Soliton Parameters:
k=
1.6 1.0
eta0=
-2.0 4.0
```

For the system mentioned above we will have 4 eigen values: $\lambda_0 = 1.494$, $\lambda_1 = 0.874$, $\lambda_2 = -0.847$ and $\lambda_3 = -1.494$. For λ_2 and λ_3 we will get the same profiles as λ_0 and λ_1 except they travel in the opposite direction.

For λ_0 the derived KdV and corresponding initial profiles are-

$$\partial_\tau f_j^{(0)} + 1.377 f_j^{(0)} \partial_\xi f_j^{(0)} - 0.083 \partial_\xi^3 f_j^{(0)} = 0$$

$$\psi_1 = \sqrt{\rho_1} e^{i \int_0^x v_1 dx'} \qquad \psi_2 = \sqrt{\rho_2} e^{i \int_0^x v_2 dx'}$$

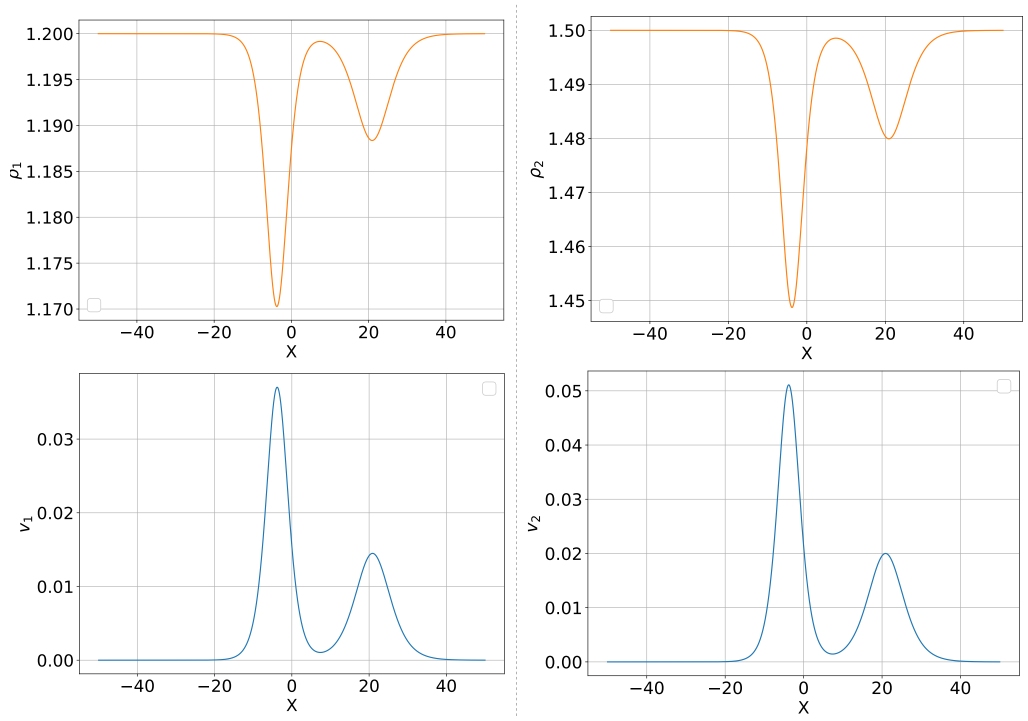


Figure 4.1: ρ_1, v_1 and ρ_2, v_2 profiles for λ_0

For λ_3 we get the same coefficient for the nonlinear term and the coefficient of dispersion term is same in magnitude but opposite in sign because of which we get the same profiles but they travel in opposite direction.

For λ_1 the derived KdV and corresponding initial profiles are-

$$\partial_\tau f_j^{(0)} - 1.377 f_j^{(0)} \partial_\xi f_j^{(0)} - 0.142 \partial_\xi^3 f_j^{(0)} = 0$$

$$\psi_1 = \sqrt{\rho_1} e^{i \int_0^x v_1 dx'} \qquad \psi_2 = \sqrt{\rho_2} e^{i \int_0^x v_2 dx'}$$

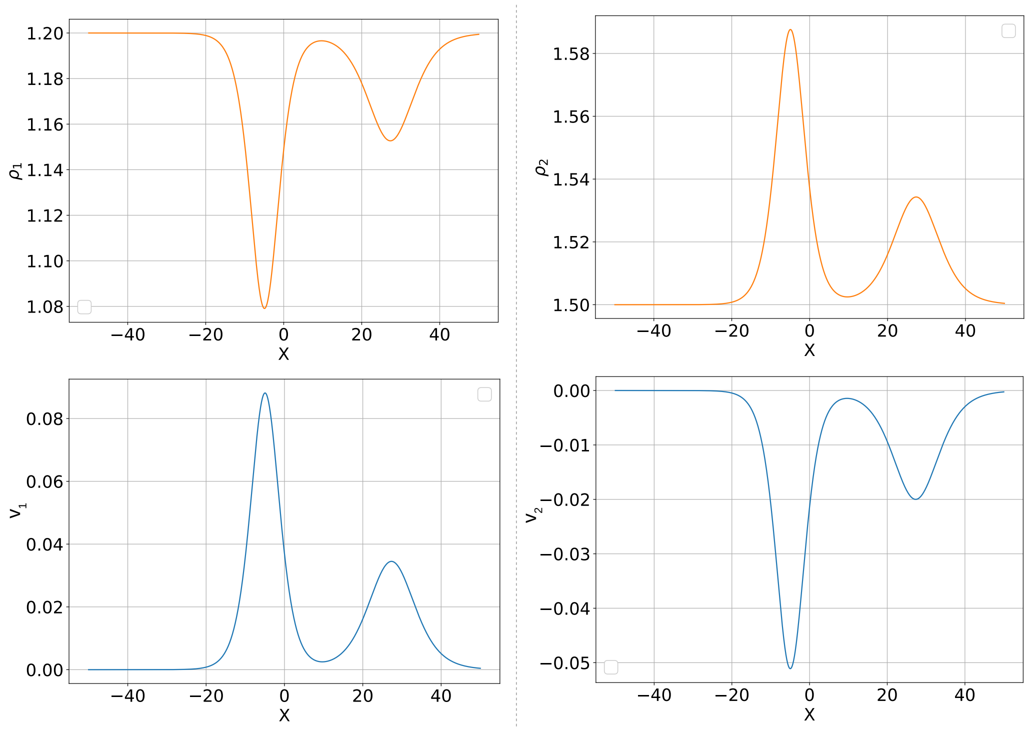


Figure 4.2: ρ_1, v_1 and ρ_2, v_2 profiles for λ_1

For λ_2 we get the same coefficient for the nonlinear term and the coefficient of dispersion term is same in magnitude but opposite in sign because of which we get the same profiles but they travel in opposite direction.

Lets see an example on how to use user defined profiles $mSol = 0$. See the code snippet below-

```
void gaussian(double *gaus,double a,double x0,double sig){
 for (int r = 0; r < gridP; r++)
   gaus[r] = a*exp(-pow((X[r]-x0),2)/(2*pow(sig,2)));
}

void numericalIntegration(double *input,double *integral){
 for (int r=0; r<gridP-1; r++) {
   if (r==0)
     integral[r]=delx*(input[r]+input[r+1])/2;
   else
     integral[r]=integral[r-1]
                +delx*(input[r]+input[r+1])/2;
 }
 integral[gridP-1]=integral[gridP-2];
}

if (mSol == 0){ //User defined profile
 double peakR[3] = {-0.0171, -0.0141, -0.0301};
 double sigmaR[3] = {3.501, 3.501, 3.501};
 double centreR[3] = {0.0, 0.0, 0.0};
 double peakV[3] = {-0.0257, -0.0266, -0.0380};
 double sigmaV[3] = {3.501, 3.501, 3.501};
 double centreV[3] = {0.0, 0.0, 0.0};

 double *baseFunc = (double*)calloc(gridP,sizeof(double));
 for (int r = 0; r < nCoup; r++) {
   gaussian(baseFunc,peakR[r],centreR[r],sigmaR[r]);
   for (int c = 0; c < gridP; c++)
     arg[c] = sqrt( rho0[r][r]+baseFunc[c] );
   gaussian(baseFunc,peakV[r],centreV[r],sigmaV[r]);
   for (int r = 0; r < gridP; r++) {
     baseFunc[r] = 0;
   }
   numericalIntegration(baseFunc,theta);
   for (int c = 0; c < gridP; c++)
     qKdV[r][c+1] = arg[c]*cos(theta[c])
                  + I*arg[c]*sin(theta[c]);
 }
 free(baseFunc);
}
```

The above code snippet is an example of how you can make any desired initial profile. For the example above we are using a gaussian profile for both the ρ and v fields for 3 component NLS. We start by defining a function *gaussian* which returns a gaussian profile over the whole domain for input amplitude, standard deviation and peak position. Remember that the domain (X),*delx* and other system defining variables are globally defined. We then specify amplitudes, standard deviation and peak positions for all the 3 ρ_k and v_k and call the function *gaussian* to get ρ_k and v_k profiles. We then integrate the v_k profiles using the function *numericalIntegration* and compose the initial profile.

Chapter 5

Numerical Time Evolution

As mentioned earlier the solver provides 3 different time stepping routines and this chapter describes each of them in details. Before starting on the particulars of each solver we first see the common routines and features shared by all.

5.1 Grid and Boundary Conditions

Fig. 5.1 shows how the computational grid is arranged.

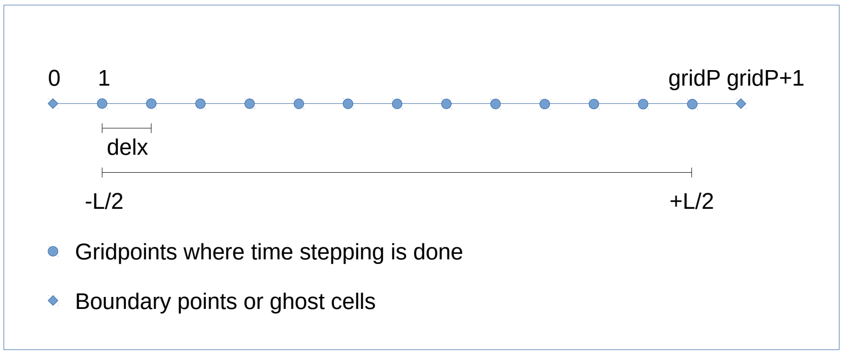

0
1
gridP gridP+1
delx
-L/2
+L/2
Gridpoints where time stepping is done
Boundary points or ghost cells

Figure 5.1: Grid

There are total $gridP + 2$ gridpoints (2 extra than the user input). These 2 extra grid points are used to implement the boundary conditions. Therefore the time stepping, using the discretized form of Eq. 1.1, is done only on the gridpoints 1 to $gridP$. You can apply any boundary condition that you wish to using the subroutine *BoundaryCondition.c*. The subroutines takes in an array and apply the desired boundary condition on it. It has three boundary conditions already coded in it-

1. Homogenous Neumann 1st order boundary condition
2. Homogenous Neumann 2nd order boundary condition
3. Periodic Boundary Condition

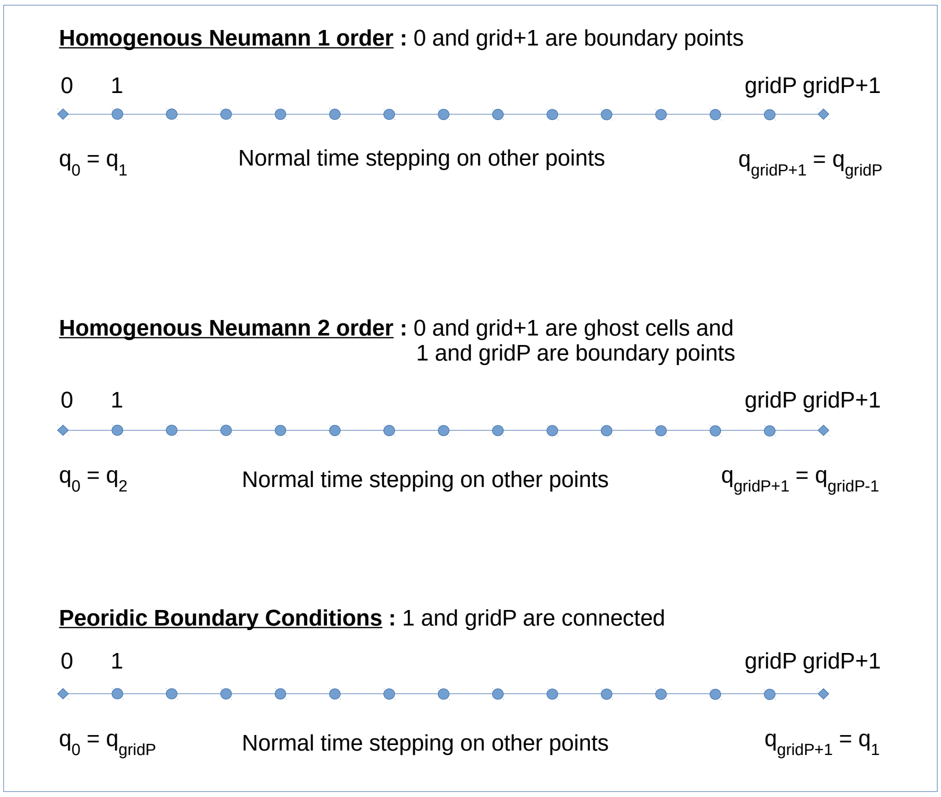

Homogenous Neumann 1 order : 0 and grid+1 are boundary points
0
1
gridP gridP+1
$q_0 = q_1$
Normal time stepping on other points
$q_{gridP+1} = q_{gridP}$
Homogenous Neumann 2 order : 0 and grid+1 are ghost cells and 1 and gridP are boundary points
0
1
gridP gridP+1
$q_0 = q_2$
Normal time stepping on other points
$q_{gridP+1} = q_{gridP-1}$
Peoridic Boundary Conditions : 1 and gridP are connected
0
1
gridP gridP+1
$q_0 = q_{gridP}$
Normal time stepping on other points
$q_{gridP+1} = q_1$

Figure 5.2: Different Boundary Conditions

5.1.1 Homogenous Neumann 1st order boundary condition

In this case the actual domain for the differential equation is from $l/2 - delx$ to $l/2 + delx$, where $gridpoint = 0$ corresponds to $l/2 - delx$ and $gridpoint = gridP + 1$ corresponds to $l/2 + delx$. The differential equation is valid on all the points from 1 to $gridP$ and 0 and $gridP + 1$ are the points where we have the boundary conditions.

The homogeneous Neumann boundary condition is implemented using forward and backward difference at points 0 and $gridP + 1$ respectively and hence the truncation error is of the order $delx$.

Boundary Condition at gridpoint 0:

$$\left.\frac{\partial q}{\partial x}\right|_{x=-l/2} \approx \frac{q_1 - q_0}{\Delta x} = 0$$

Boundary Condition at gridpoint $gridP + 1$:

$$\left.\frac{\partial q}{\partial x}\right|_{x=+l/2} \approx \frac{q_{gridP+1} - q_{gridP}}{\Delta x} = 0$$

5.1.2 Homogenous Neumann 2^{nd} order boundary condition

In this case the actual domain for the differential equation is from $l/2$ to $l/2$, where $gridpoint = 1$ corresponds to $l/2$ and $gridpoint = gridP$ corresponds to $l/2$. The differential equation is valid on all the points from 2 to $gridP - 1$ and 1 and $gridP$ are the points where we have the boundary conditions. $gridpoint = 0$ and $gridpoint = gridP + 1$ are ghost cells used to implement the second order accurate boundary condition.

The homogeneous Neumann boundary condition is implemented using central difference at points 1 and $gridP$. The central difference requires an extra point on the left of left boundary and on the right of right boundary and hence we have the ghost cells.

Boundary Condition at gridpoint 1:

$$\left.\frac{\partial q}{\partial x}\right|_{x=-l/2} \approx \frac{q_2 - q_0}{\Delta x} = 0$$

Boundary Condition at gridpoint $griP$:

$$\left.\frac{\partial q}{\partial x}\right|_{x=+l/2} \approx \frac{q_{gridP+1} - q_{gridP-1}}{\Delta x} = 0$$

5.1.3 Periodic boundary condition

Periodic boundary condition assumes that gridpoints 1 and gridpoints $gridP$ are connected. This means

Boundary Condition at gridpoint 1:

$$q_0 = q_{gridP}$$

Boundary Condition at gridpoint $griP$:

$$q_{gridP+1} = q_1$$

5.2 Conserved Quantities

There are three conserved quantities in system.

1. The total mass of each component is conserved.

$$\int |\psi_k|^2 dx \qquad \forall k \tag{5.1}$$

   which is numerically integrated as

$$\sum_i \frac{|\psi_k^{i+1}|^2 + |\psi_k^i|^2}{2} \Delta x \qquad \forall k$$

2. The total momentum of the system is conserved

$$\int \sum_{k=1}^{N} Im(\psi_k^* \partial_x \psi_k) dx \tag{5.2}$$

   which is numerically integrated as

$$\sum_i \sum_k \frac{1}{2} \Bigg\{ Im \left[\psi_k^{*i+1} \frac{\psi_k^{i+1+1} - \psi_k^{i+1}}{\Delta x} \right] + Im \left[\psi_k^{*i} \frac{\psi_k^{i+1} - \psi_k^{i}}{\Delta x} \right] \Bigg\} \Delta x$$

3. The hamiltonian of the system

$$\int \sum_{k=1}^{N} \left(\frac{|\partial_x \psi_k|^2}{2} + \sum_{j=1}^{N} \frac{\alpha_{kj}}{2} |\psi_k|^2 |\psi_j|^2 + \frac{G_k}{2} |\psi_k|^4 |\psi_k|^2 \right) dx \tag{5.3}$$

   which is numerically integrated as

$$\sum_i \sum_k \frac{1}{2} \Bigg[\frac{1}{2} \left| \frac{\psi_k^{i+1} - \psi_k^i}{\Delta x} \right|^2 + \sum_{j=1}^{N} \frac{\alpha_{kj}}{2} |\psi_k^i|^2 |\psi_j^i|^2 + \frac{G_k}{2} |\psi_k^i|^6$$
$$+ \frac{1}{2} \left| \frac{\psi_k^{i+1+1} - \psi_k^{i+1}}{\Delta x} \right|^2 + \sum_{j=1}^{N} \frac{\alpha_{kj}}{2} |\psi_k^{i+1}|^2 |\psi_j^{i+1}|^2 + \frac{G_k}{2} |\psi_k^{i+1}|^6 \Bigg] \Delta x$$

5.2.1 RHSFunction.c

The subroutine *RHSFunction.c* contains the discretized RHS of Eq. 1.1. The subroutine is used by the explicit time stepping schemes where a known array q is passed to the function, along with the time corresponding to q and the subroutine returns the value of discretized equation at all the points in the domain.

The RHS the Eq. 1.1 is discretized as

$$\frac{\partial q_k^m}{\partial t} = \frac{1}{i}\left[\frac{1}{2}\frac{q_k^{m+1} - 2q_k^m + q_k^{m-1}}{\Delta x^2} + \sum_{j=1}^{N} \alpha_{lj}|q_j^m|^2 q_k^m + G_{ll}|q_k^m|^4 q_k^m\right] \tag{5.4}$$

where k represents the kth component of NLS and m represents any point on the spatial grid.

If you want to use the package for some other form of Eq. 1.1 you can either scale your equation and reduce it the form of Eq. 1.1 or change the subroutine *RHSFunction.c*.

NOTE- It is very important that if you change the RHS of Eq. 1.1 in subroutine *RHSFunction.c* you also make the following changes-

1. In the subroutine *getKdVsystem* change the arrays RR, RV, VR and VV which corresponds to the coefficients of the linearised system.

2. In the subroutine *getKdVsystem* change the arrays $N1$ and $N2$ which represents nonlinear and the dispersion term in the Eq. 1.1.

3. In the subroutine *conserverQuantityCalculator* change the equation of hamiltonian to the new hamiltonian.

5.3 Leap Frog Method

The discretized equation for Leap frog looks like this

$$\frac{q_k^{m,t+\Delta t} - q_k^{m,t-\Delta t}}{\Delta t} = \frac{1}{i}\left[\frac{1}{2}\frac{q_k^{m+1,t} - 2q_k^{m,t} + q_k^{m-1,t}}{\Delta x^2} + \sum_{j=1}^{N} \alpha_{lj}|q_j^{m,t}|^2 q_k^{m,t} + G_{ll}|q_k^{m,t}|^4 q_k^{m,t}\right] \tag{5.5}$$

where $q_k^{m,t}$ value of ψ for k^{th} component at gridpoint m at time t.

This method is linearly stable for $\Delta t/(\Delta x)^2 \leq 1/4$ and the truncation error of the scheme is of the order $O((\Delta t)^2) + O((\Delta x)^2)$

The complete algorithm is described in the Fig. 5.3

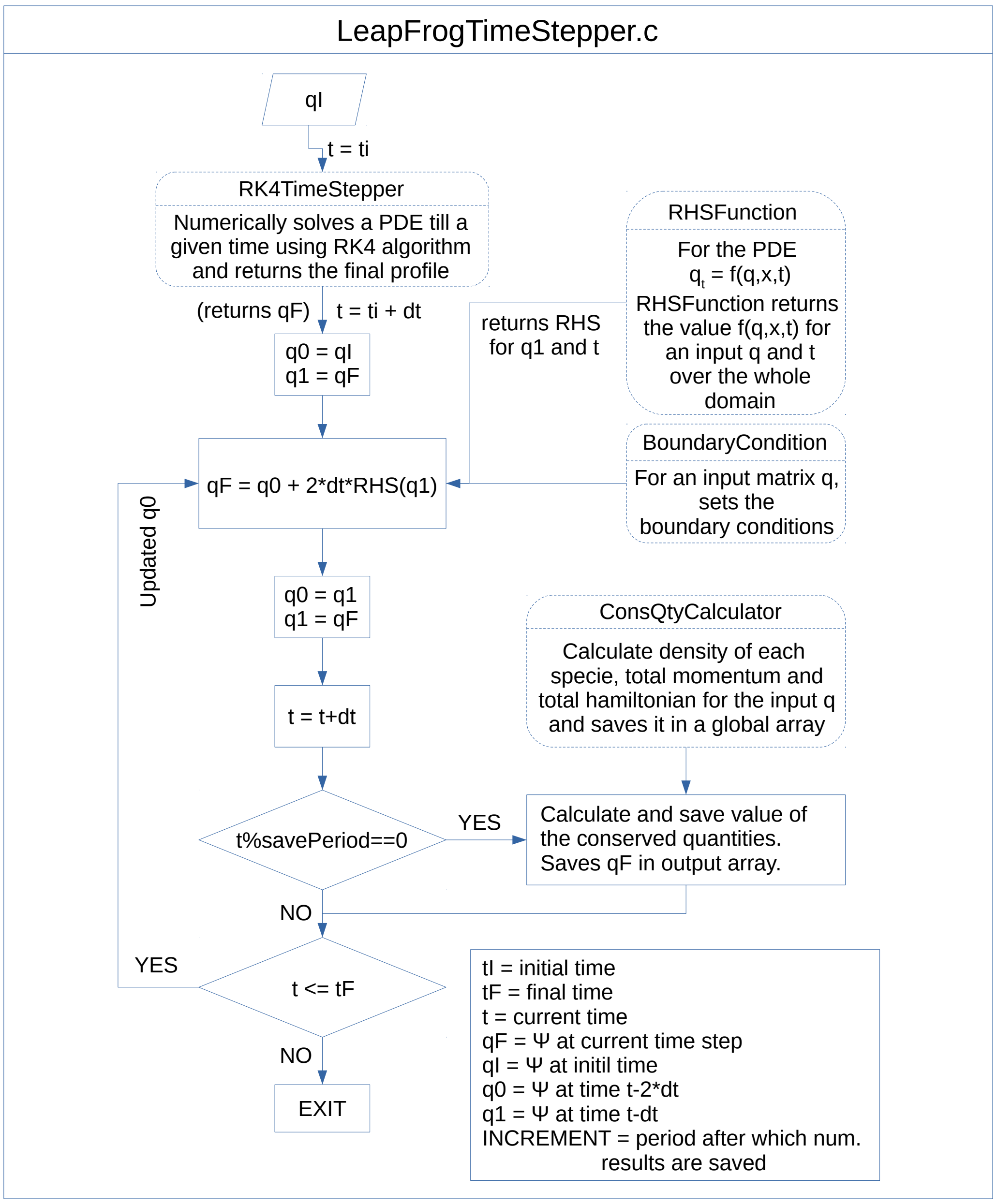

LeapFrogTimeStepper.c
qI
t = ti
RK4TimeStepper
Numerically solves a PDE till a given time using RK4 algorithm and returns the final profile
(returns qF)
t = ti + dt
q0 = qI
q1 = qF
qF = q0 + 2*dt*RHS(q1)
Updated q0
returns RHS for q1 and t
RHSFunction
For the PDE $q_t = f(q,x,t)$ RHSFunction returns the value f(q,x,t) for an input q and t over the whole domain
BoundaryCondition
For an input matrix q, sets the boundary conditions
q0 = q1
q1 = qF
t = t+dt
ConsQtyCalculator
Calculate density of each specie, total momentum and total hamiltonian for the input q and saves it in a global array
t%savePeriod==0
YES
Calculate and save value of the conserved quantities. Saves qF in output array.
NO
YES
t <= tF
NO
EXIT
tI = initial time
tF = final time
t = current time
qF = Ψ at current time step
qI = Ψ at initil time
q0 = Ψ at time t-2*dt
q1 = Ψ at time t-dt
INCREMENT = period after which num. results are saved

Figure 5.3: Leap Frog Time Stepper

5.4 Runge Kutta 4th Order Explicit Method

As stated earlier this subroutine also uses the *RHSFunction.c* subroutine to calculate the RHS for time stepping. RK4 method uses weighted average of 4 different RHS values. These different RHS values and the complete algorithm is described in Fig. 5.4

$$k1_k^{m,t} = \frac{1}{i}\left[\frac{1}{2}\frac{q_k^{m+1,t} - 2q_k^{m,t} + q_k^{m-1,t}}{\Delta x^2} + \sum_{j=1}^{N} \alpha_{lj}|q_j^{m,t}|^2 q_k^{m,t} + G_{ll}|q_k^{m,t}|^4 q_k^{m,t}\right]$$

$$qE_k^{m,t+\Delta t/2} = q_k^{m,t} + \frac{\Delta t}{2} k1_k^{m,t}$$

$$k2_k^{m,t+\Delta t/2} = \frac{1}{i}\left[\frac{1}{2}\frac{qE_k^{m+1,t+\Delta t/2} - 2qE_k^{m,t+\Delta t/2} + qE_k^{m-1,t+\Delta t/2}}{(\Delta x)^2} + \right.$$
$$\left. \sum_{j=1}^{N} \alpha_{lj}|qE_j^{m,t+\Delta t/2}|^2 qE_k^{m,t+\Delta t/2} + G_{ll}|qE_k^{m,t+\Delta t/2}|^4 qE_k^{m,t+\Delta t/2}\right]$$

$$qI_k^{m,t+\Delta t/2} = q_k^{m,t} + \frac{\Delta t}{2} k2_k^{m,t+\Delta t/2}$$

$$k3_k^{m,t+\Delta t/2} = \frac{1}{i}\left[\frac{1}{2}\frac{qI_k^{m+1,t+\Delta t/2} - 2qI_k^{m,t+\Delta t/2} + qI_k^{m-1,t+\Delta t/2}}{(\Delta x)^2} + \right.$$
$$\left. \sum_{j=1}^{N} \alpha_{lj}|qI_j^{m,t+\Delta t/2}|^2 qI_k^{m,t+\Delta t/2} + G_{ll}|qI_k^{m,t+\Delta t/2}|^4 qI_k^{m,t+\Delta t/2}\right]$$

$$qLF_k^{m,t+\Delta t/2} = q_k^{m,t} + \Delta t k3_k^{m,t+\Delta t/2}$$

$$k4_k^{m,t+\Delta} = \frac{1}{i}\left[\frac{1}{2}\frac{qLF_k^{m+1,t+\Delta t/2} - 2qLF_k^{m,t+\Delta t/2} + qLF_k^{m-1,t+\Delta t/2}}{(\Delta x)^2} + \right.$$
$$\left. \sum_{j=1}^{N} \alpha_{lj}|qLF_j^{m,t+\Delta t/2}|^2 qLF_k^{m,t+\Delta t/2} + G_{ll}|qLF_k^{m,t+\Delta t/2}|^4 qLF_k^{m,t+\Delta t/2}\right]$$

$$q_k^{m,t+\Delta t} = q_k^{m,t} + \Delta t k4_k^{m,t+\Delta}$$

The truncation error of the scheme is of the order $O((\Delta t)^4) + O((\Delta x)^2)$

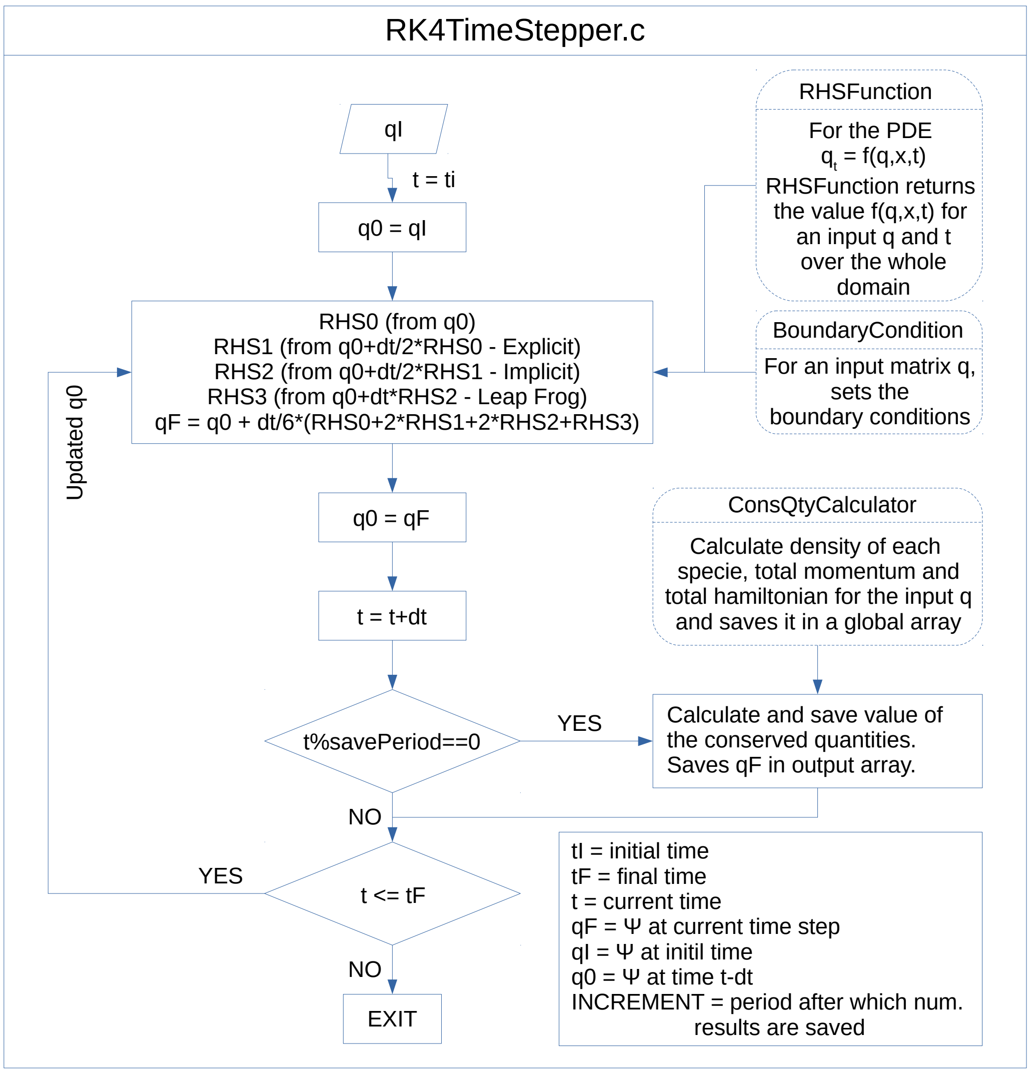

RK4TimeStepper.c
qI
t = ti
q0 = qI
RHS0 (from q0)
RHS1 (from q0+dt/2*RHS0 - Explicit)
RHS2 (from q0+dt/2*RHS1 - Implicit)
RHS3 (from q0+dt*RHS2 - Leap Frog)
qF = q0 + dt/6*(RHS0+2*RHS1+2*RHS2+RHS3)
RHSFunction
For the PDE
q_t = f(q,x,t)
RHSFunction returns the value f(q,x,t) for an input q and t over the whole domain
BoundaryCondition
For an input matrix q, sets the boundary conditions
Updated q0
q0 = qF
t = t+dt
ConsQtyCalculator
Calculate density of each specie, total momentum and total hamiltonian for the input q and saves it in a global array
t%savePeriod==0
YES
Calculate and save value of the conserved quantities. Saves qF in output array.
NO
YES
t <= tF
NO
EXIT
tI = initial time
tF = final time
t = current time
qF = Ψ at current time step
qI = Ψ at initil time
q0 = Ψ at time t-dt
INCREMENT = period after which num. results are saved

Figure 5.4: RK4 Time Stepper

5.5 Crank Nicolson Implicit scheme

Crank Nicolson is an implicit scheme and Eq. 1.1 is a nonlinear PDE which means that after discretization we will get a system of nonlinear equations. To solve this system of nonlinear equations we use the Picards iteration technique. The discretization of the PDE is done according to Crank Nicolson Scheme.

Discretization

$$
\begin{aligned}
i\frac{q_k^{m,t+1}-q_k^{m,t}}{\Delta t} &= \\
\frac{-1}{2}\frac{1}{2\Delta x^2}&\left[q_k^{m+1,t+1}-2q_k^{m,t+1}+q_k^{m-1,t+1}+q_k^{m+1,t}-2q_k^{m,t}+q_k^{m-1,t}\right]+ \\
&\frac{1}{2}\left[\sum_{j=1}^{N}\alpha_{lj}|q_j^{m,t+1}|^2q_k^{m,t+1}+\sum_{j=1}^{N}\alpha_{lj}|q_j^{m,t}|^2q_k^{m,t}\right]+ \\
&\frac{1}{2}\left[G_{ll}|q_k^{m,t+1}|^4q_k^{m,t+1}+G_{ll}|q_k^{m,t]}|^4q_k^{m,t}\right] \quad (5.6)
\end{aligned}
$$

which after simplification becomes

$$
\begin{aligned}
q_k^{m,t+1}\left(i-\frac{\Delta t}{2\Delta x^2}\right)+q_k^{m+1,t+1}\left(\frac{\Delta t}{4\Delta x^2}\right)+q_k^{m-1,t+1}\left(\frac{\Delta t}{4\Delta x^2}\right) &= \\
q_k^{m,t}\left(i+\frac{\Delta t}{2\Delta x^2}\right)-q_k^{m+1,t}\left(\frac{\Delta t}{4\Delta x^2}\right)-q_k^{m-1,t}\left(\frac{\Delta t}{4\Delta x^2}\right)&+ \\
\frac{1}{2}\left[\sum_{j=1}^{N}\alpha_{lj}|\tilde{q_j}^{m,t+1}|^2\tilde{q_k}^{m,t+1}+\sum_{j=1}^{N}\alpha_{lj}|q_j^{m,t}|^2q_k^{m,t}\right]&+ \\
\frac{1}{2}\left[G_{ll}|\tilde{q_k}^{m,t+1}|^4\tilde{q_k}^{m,t+1}+G_{ll}|q_k^{m,t]}|^4q_k^{m,t}\right]& \quad (5.7)
\end{aligned}
$$

where $\tilde{q_k}$ is an approximation for q_k. Since we are using Picards iteration to solve the nonlinearity at every time step we approximate the nonlinearity with the values of the previous time step. We then solve the linear system after assuming $\tilde{q_k}$ and then update $\tilde{q_k}$ with the solution of the linear system. We continue this till the change in 2 consecutive values is not greater than a desired value.

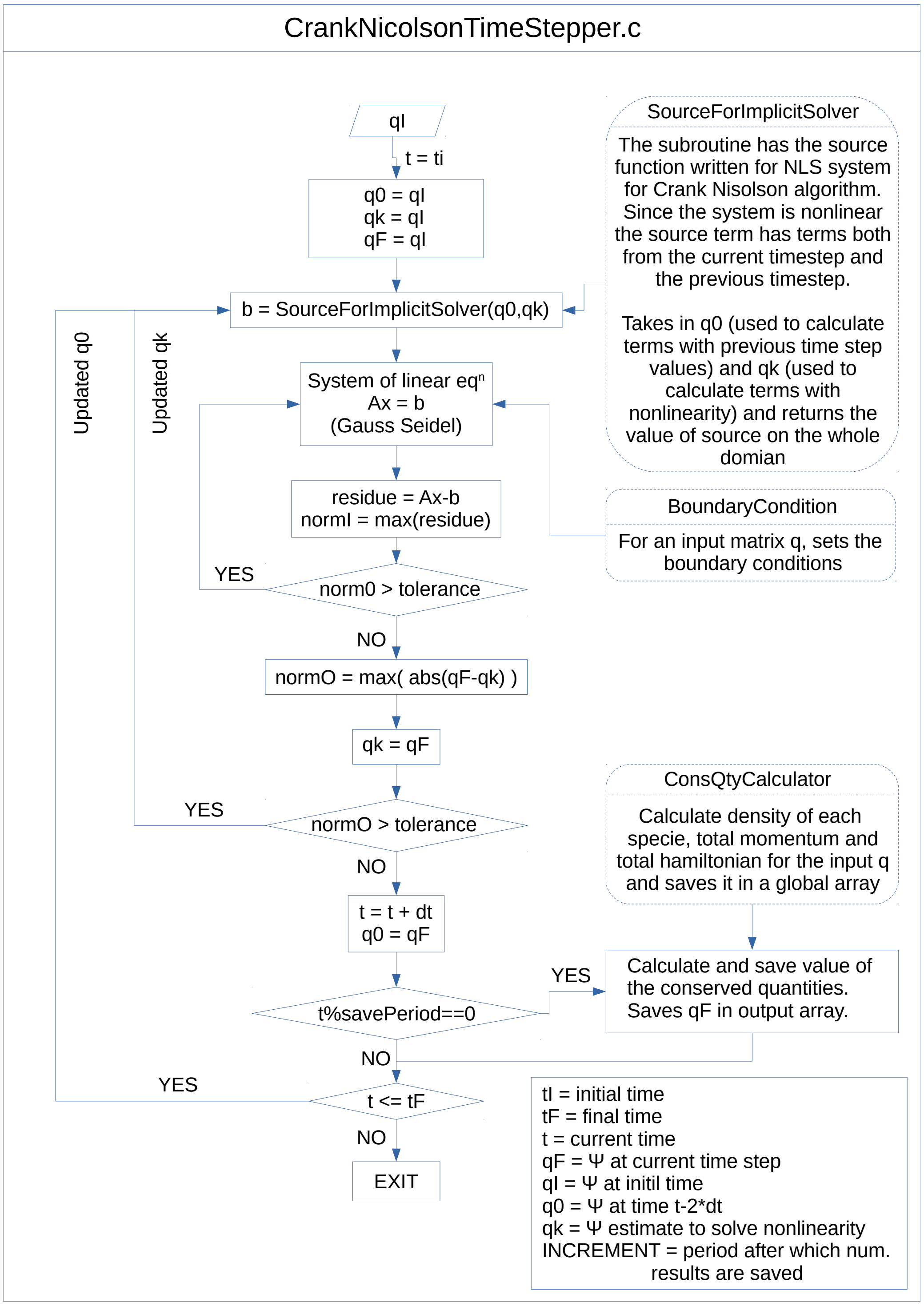

CrankNicolsonTimeStepper.c
qI
t = ti
q0 = qI
qk = qI
qF = qI
b = SourceForImplicitSolver(q0,qk)
System of linear eqⁿ
Ax = b
(Gauss Seidel)
residue = Ax-b
normI = max(residue)
YES
norm0 > tolerance
NO
normO = max(abs(qF-qk))
qk = qF
YES
normO > tolerance
NO
t = t + dt
q0 = qF
YES
t%savePeriod==0
NO
YES
t <= tF
NO
EXIT
Updated q0
Updated qk
SourceForImplicitSolver
The subroutine has the source function written for NLS system for Crank Nisolson algorithm. Since the system is nonlinear the source term has terms both from the current timestep and the previous timestep.
Takes in q0 (used to calculate terms with previous time step values) and qk (used to calculate terms with nonlinearity) and returns the value of source on the whole domian
BoundaryCondition
For an input matrix q, sets the boundary conditions
ConsQtyCalculator
Calculate density of each specie, total momentum and total hamiltonian for the input q and saves it in a global array
Calculate and save value of the conserved quantities. Saves qF in output array.
tI = initial time
tF = final time
t = current time
qF = Ψ at current time step
qI = Ψ at initil time
q0 = Ψ at time t-2*dt
qk = Ψ estimate to solve nonlinearity
INCREMENT = period after which num. results are saved

Figure 5.5: Crank Nicolson Time Stepper

Chapter 6

Post Processing

The output binary files are read by the python notebook *PostProcessing.ipynb* and the notebook creates 5 numpy arrays of the data -

1. psiNum - A $gridP \times (tf/savePeriod + 1)nCoup$ array which contains the numerical results (KdV soliton evolved using NLS dynamics) for all components at all output time levels.

2. psiAna - A $gridP \times (tf/savePeriod + 1)nCoup$ array which contains ψ created from KdV solitons for all components at all output time levels.

3. consQty1 - A $nCoup \times tf/savePeriod + 1$ array which contains the value of under area of ρ for all components at all time levels.

4. consQty2 - A $1 \times tf/savePeriod + 1$ array which contains the value of total momentum of the system at all time levels.

5. consQty3 - A $1 \times tf/savePeriod + 1$ array which contains the value of total hamiltonian of the system at all time levels.

See Fig. 6.1 to understand how the data is stored in the arrays.

We then create the following 4 variables for ρ_k and v_k from $psiNum$ and $psiAna$ - $rhoNum$, $velNum$ from $psiNum$ and $rhoAna$, $velAna$ from $psiAna$. The data in all these variables are stored in the same way as in $psiNum$ and $psiAna$.

The *PostProcessing.ipynb* notebook also plots the relative change in the conserved quantities and tracks the peak of the soliton like solution we get from using this approach.

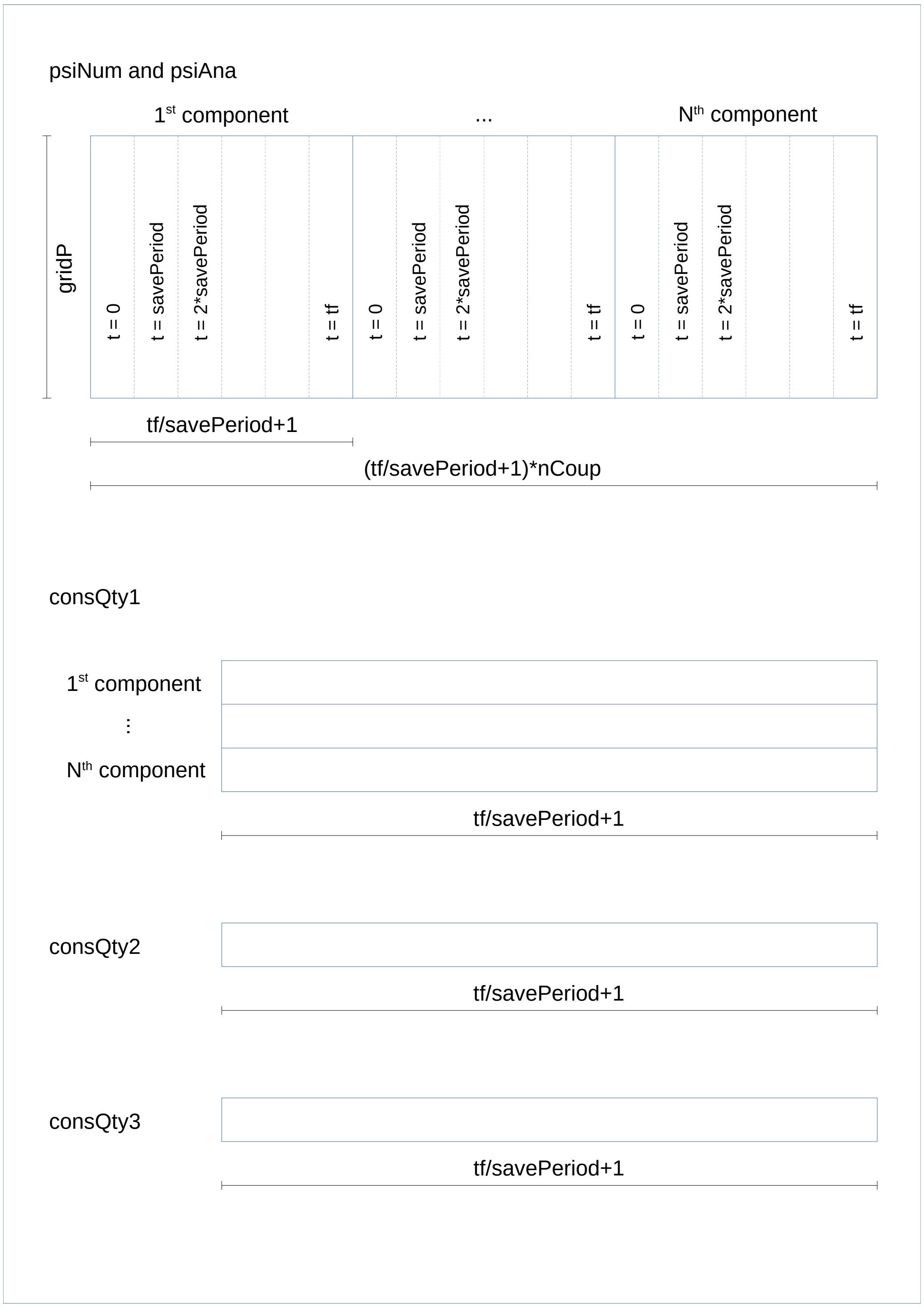


Figure 6.1: Output data of the Simulation

Chapter 7

Simulation Results

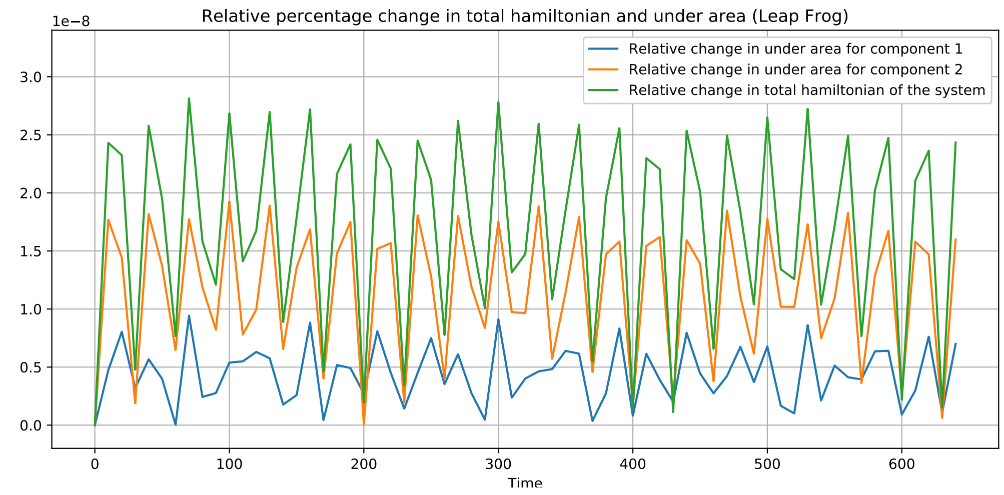
The chapter discusses some simulation results from different schemes and features in the output data.

7.1 Simulation Features

The following features do not depend on the input system

1. Since the initial conditions that we provide are not exactly NLS solitons, the localized bump/dip releases small oscillations (called radiations) as it evolves as seen on the left hand side of the bump in fig. 1.5.

2. The Leap Frog time stepping scheme inherently gives oscillating results. It is due to the fact that the first time step is calculated with some other scheme, RK4 in our case. These oscillation can be clearly seen in the conserved quantities (1.2). Because of this nature of the numerical scheme you also see the whole ρ and v profile shift up and down. This is completely a numerical artifact since these oscillation are absent in Runge Kutta 4^{th} order scheme and Crank Nicolson Scheme as seen in fig. 7.1. The Leap Frog Scheme is a more than 2 times faster than RK4 scheme but it comes with the trade off of these oscillations.



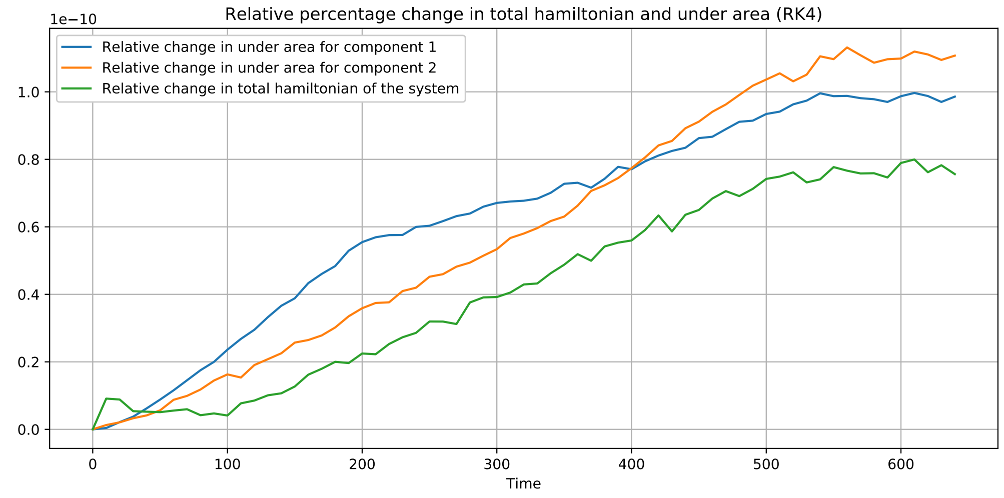


Figure 7.1: Oscillations in Leap Frog Scheme

3. There is a sudden change in the total momentum of the system whenever any small radiation hits the boundary with homogeneous Neumann boundary condition as seen in fig. 7.2.The radiation after hitting the boundary reflects back and remains inside the domain. This is because of the fact that no flux boundary condition on ψ does not mean no flux boundary condition in ρ and v.

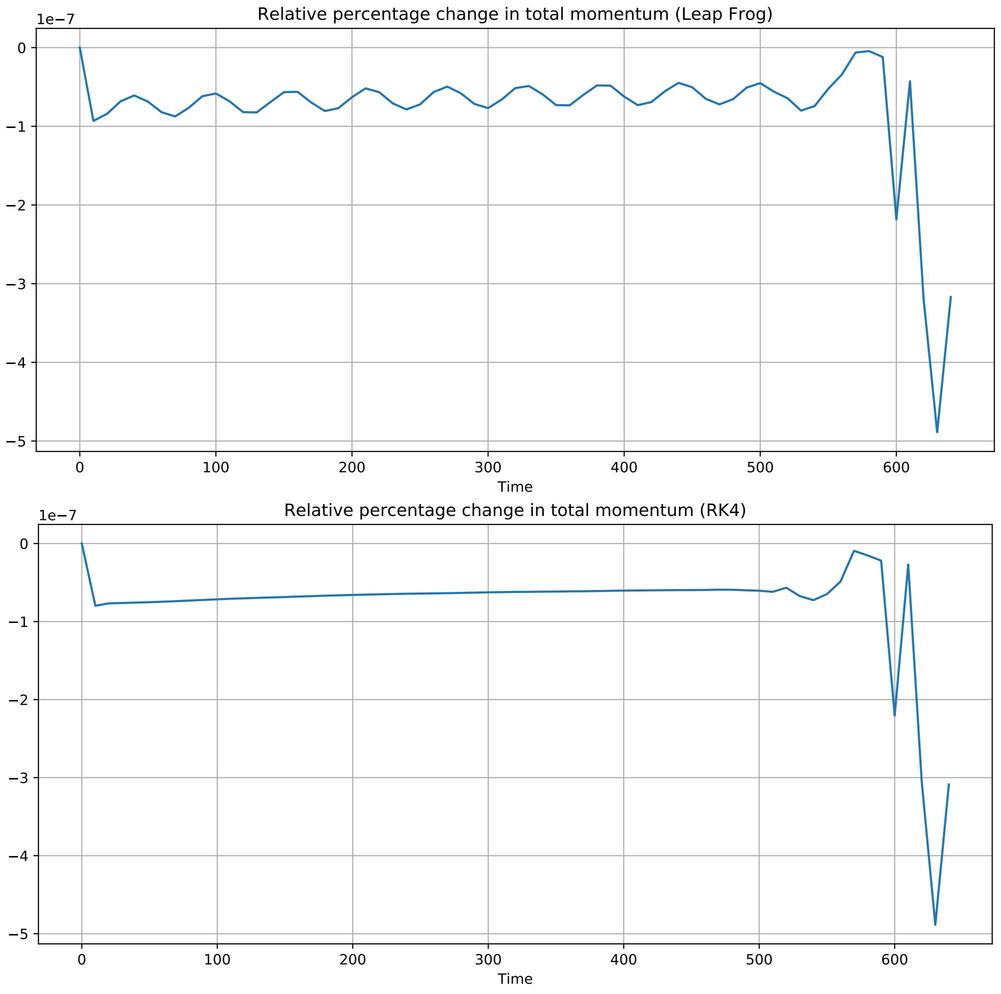


Figure 7.2: Momentum change when radiations hit the boundary

4. The soliton like initial conditions travel in straight line. Fig. 7.3 shows the position vs time graph for all the peaks/bumps for a 2 NLS 3 soliton system.

```
N-COUPLED = 2
M-SOLITON = 3
lambda = 0
epsilon = 0.200
k[0] = 0.000 + 1.100i
k[1] = 0.000 + 1.400i
k[2] = 0.000 + 1.800i
```

```
eta0[0] = 0.000
eta0[1] = 1.800
eta0[2] = 2.400
Alpha Matrix
1.000  0.500
0.500  1.200
Rho0 Matrix
1.200  0.000
0.000  1.400
```

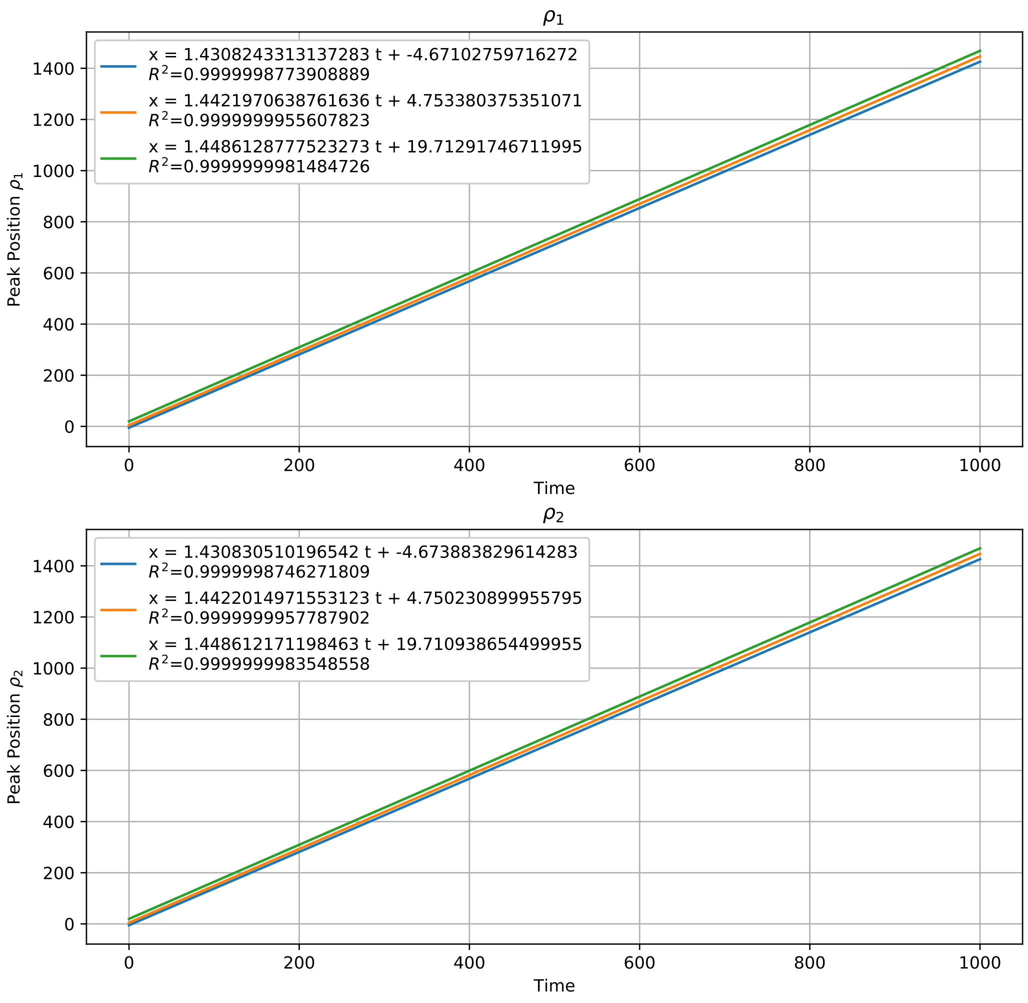


Figure 7.3: Position of traveling peak/bump

5. For 1 soliton initial conditions in all the cases that we tested the bump/peak reaches an almost equilibrium value and just oscillates about a mean value as shown in following figures.

```
N-COUPLED = 2
M-SOLITON = 1
lambda = 0
epsilon = 0.200
k = 0.000 + 1.118i
eta0 = 0.000
Alpha Matrix
1.000  0.500
0.500  1.200
Rho0 Matrix
1.200  0.000
0.000  1.400
```

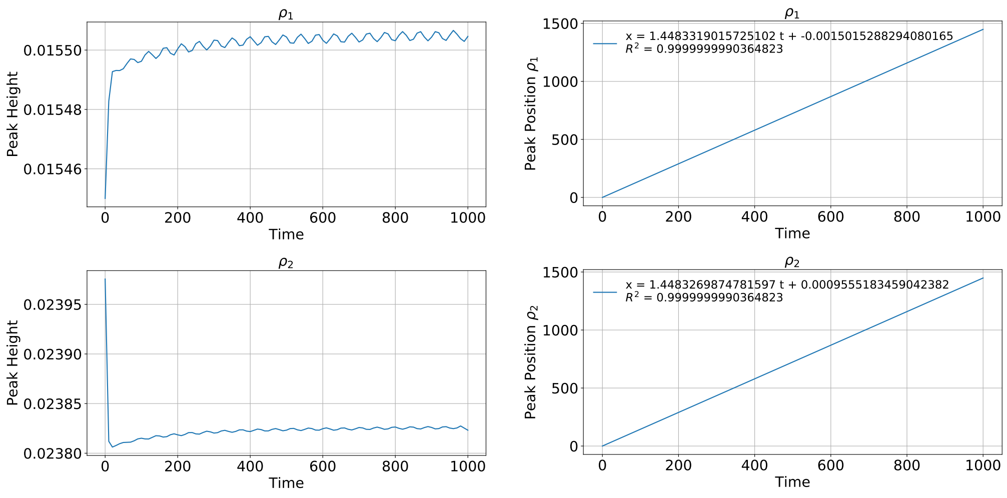


Figure 7.4: Position of traveling and height of peak/bump

```
N-COUPLED = 3
M-SOLITON = 1
lambda = 0
epsilon = 0.200
k = 0.000 + 1.118i
eta0 = 0.000
```

```
Alpha Matrix
1.000  0.500  0.500
0.500  1.200  0.500
0.500  0.500  1.500
Rho0 Matrix
1.200  0.000  0.000
0.000  1.400  0.000
0.000  0.000  1.500
```

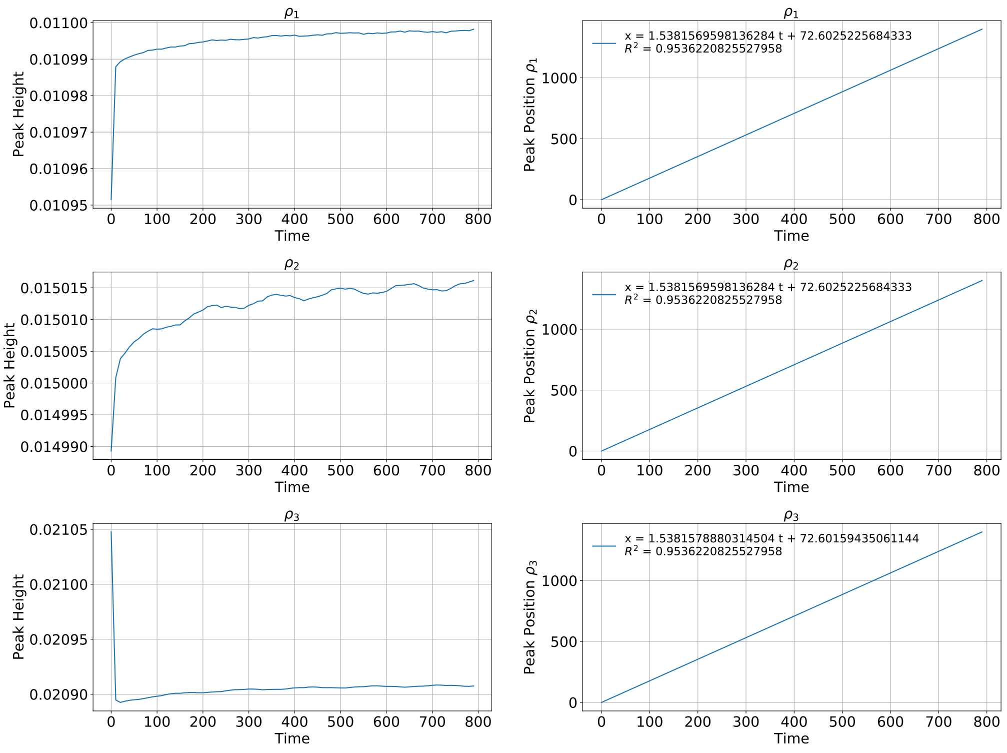


Figure 7.5: Position of traveling and height of peak/bump

Bibliography

[1] E. Madelung. Die lorentz-invarianz der diracschen theorie in vektorform. *Zeitschrift für Physik*, 57(9):573–574, Sep 1929.

[2] Swetlana Swarup, Vishal Vasan, and Manas Kulkarni. Provable bounds for the korteweg–de vries reduction in multi-component nonlinear schrödinger equation. *Journal of Physics A*, 53:135206, 2020.