2012

Study of Vortex Ring Dynamics in the Nonlinear Schrödinger Equation Utilizing GPU-Accelerated High-Order Compact Numerical Integrators

Ronald Meyer Caplan
Claremont Graduate University

DOI: 10.5642/cguetd/52

STUDY OF VORTEX RING DYNAMICS IN THE NONLINEAR SCHRÖDINGER EQUATION UTILIZING GPU-ACCELERATED HIGH-ORDER COMPACT NUMERICAL INTEGRATORS

A Dissertation

Presented to the Faculty of

Claremont Graduate University

and

San Diego State University

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

in

Computational Science

---

by

Ronald Meyer Caplan

May 2012

APPROVAL OF THE REVIEW COMMITTEE

This dissertation has been duly read, reviewed, and critiqued by the Committee listed below, which hereby approves the manuscript of

Ronald Meyer Caplan

as fulfilling the scope and quality requirements for meriting the degree of

Doctor of Philosophy.

Ricardo Carretero-González, Chair
Department of Mathematics and Statistics, San Diego State University
Professor

Peter Blomgren
Department of Mathematics and Statistics, San Diego State University
Assistant Professor

Michael Bromley
Department of Physics, San Diego State University
Adjunct Professor

Ali Nadim
Department of Mathematics, Claremont Graduate University
Professor

Allon Percus
Department of Mathematics, Claremont Graduate University
Associate Professor

ABSTRACT

Study of Vortex Ring Dynamics in the
Nonlinear Schrödinger Equation utilizing
GPU-Accelerated High-Order Compact Numerical Integrators
by
Ronald Meyer Caplan
Claremont Graduate University and San Diego State University, 2012

We numerically study the dynamics and interactions of vortex rings in the nonlinear Schrödinger equation (NLSE). Single ring dynamics for both bright and dark vortex rings are explored including their traverse velocity, stability, and perturbations resulting in quadrupole oscillations. Multi-ring dynamics of dark vortex rings are investigated, including scattering and merging of two colliding rings, leapfrogging interactions of co-traveling rings, as well as co-moving steady-state multi-ring ensembles. Simulations of choreographed multi-ring setups are also performed, leading to intriguing interaction dynamics.

Due to the inherent lack of a closed form solution for vortex rings and the dimensionality where they live, efficient numerical methods to integrate the NLSE have to be developed in order to perform the extensive number of required simulations. To facilitate this, compact high-order numerical schemes for the spatial derivatives are developed which include a new semi-compact modulus-squared Dirichlet boundary condition. The schemes are combined with a fourth-order Runge-Kutta time-stepping scheme in order to keep the overall method fully explicit. To ensure efficient use of the schemes, a stability analysis is performed to find bounds on the largest usable time step-size as a function of the spatial step-size.

The numerical methods are implemented into codes which are run on NVIDIA graphic processing unit (GPU) parallel architectures. The codes running on the GPU are shown to be many times faster than their serial counterparts. The codes are developed with

future usability in mind, and therefore are written to interface with MATLAB utilizing custom GPU-enabled C codes with a MEX-compiler interface. The codes are combined into a software package called NLSEmagic which is freely distributed on a dedicated website allowing reproducibility of the simulations presented in the study.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Prof. Ricardo Carretero who took time out of his busy schedule to give me extensive guidance and help with a multitude of topics which are too numerous to mention. I would also like to thank Profs. Michael Bromley, Peter Blomgren, Allon Percus, and Ali Nadim for their time in reviewing this thesis and for their past course instruction.

Throughout the research, additional help was received from other professors in the field. I want to thank Prof. Juan Belmonte for the providing the main idea used in Appendix B, and Prof. Paul Roberts and Prof. Herbert Levine for their helpful discussions.

I would also like to acknowledge the help of my colleagues in the Computational Science Research Center Laboratory who often assisted me with problems, including Mohammad Abouali for his suggestion in optimizing the GPU codes.

I wish to acknowledge the Computational Science Research Center for their assistance and generous financial support throughout my studies at SDSU and CGU, as well as the partial support from the National Science Foundation.

Last but certainly not least, I would like to thank my wife Molly for her great patience, understanding and support throughout the writing of this thesis.

TABLE OF CONTENTS

PAGE

LIST OF TABLES

PAGE

LIST OF FIGURES

PAGE

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

1.1.1 The Nonlinear Schrödinger Equation

The nonlinear Schrödinger equation (NLSE) is a universal model describing the evolution and propagation of complex field envelopes in nonlinear dispersive media. As such, it is used to describe many physical systems including the evolution of water waves, nonlinear optics, thermodynamic pulses, nonlinear waves in fluid dynamics, and waves in semiconductors [1–3].

One of the most relevant applications with regards to this dissertation is the simulation of the dynamics of Bose-Einstein condensates (BECs) of dilute gases [4]. BECs are a state of matter where a collection of bosons, whose de Broglie wavelengths become comparable to their mean separation distance when cooled to extremely low temperatures, undergo a phase transition so that a significant fraction of the bosons find themselves in the same quantum state. In such a case, the collection of bosons behave as one large near-macroscopic coherent particle. In the case of dilute gases, a BEC is formed by cooling the gas to extremely low temperatures (on the order of 10^{-7}K). In such a BEC, the gas cloud behaves as one very large, near-macroscopic atom which can be described by a mean-field approximation (where only two-body collisions are considered, and such collisions are considered to be felt only locally) by the NLSE with appropriate parameters. BECs have also been realized in other bosonic systems such as magnons in ferromagnetic magnets [5], polaritons in semiconductors [6], and photons within an optical cavity [7].

BECs are described as either ‘attractive’ or ‘repulsive’. Attractive BECs are those in which the scattering length for the interaction of atoms in the BEC is negative. This can often lead to the ‘collapse’ of the BEC, where the density condenses into a small area rapidly and then explode outwards similar to a supernova (in fact, such collapse in a BEC has been dubbed a ‘Bosenova’ [8–11]). Due to the unstable nature of attractive BECs, most experiments use repulsive BECs where the scattering length of interacting atoms is positive. In such a case, the BEC expands and dilutes.

The mean-field dynamics of the BEC can be described using the Gross-Pitaevskii (GP) equation which is a NLSE with an added external potential term. The GP equation for the wavefunction (Ψ) for a three-dimensional BEC as described in Ref. [12] is defined as:

$$i\hbar\frac{\partial\Psi}{\partial t}+\frac{\hbar^2}{2m_a}\nabla^2\Psi-V_{\text{ext}}(\mathbf{r})\Psi-\frac{4\pi\hbar^2a_0}{m_a}|\Psi|^2\Psi=0, \tag{1.1}$$

where $\hbar$ is the reduced Planck constant, m_a is the mass of one of the atoms in the condensate, $V_{\text{ext}}(\mathbf{r})$ is the external potential function, $\nabla^2\Psi$ is the three-dimensional Laplacian, and a_0 is the s-wave scattering length. In an attractive BEC, $a_0<0$, while in a repulsive BEC, $a_0>0$. We note here that using the GP equation to model the collapse of an attractive BEC predicts that the BEC will collapse into an infinite density singularity in finite time which does not capture the details of the actual BEC dynamics that eventually “explodes back out” after the initial collapse (the Bosenova phenomenon) [10].

While a large amount of current research using the NLSE is in the realm of BECs (the central application of this work), in order to allow for a more general study of the relevant dynamics and structures, we use a non-dimensionalized form of the NLSE. This allows the results and codes developed from this study to be directly relevant to a much wider group of researchers, increasing its impact. The form of the NLSE we use is

$$i\,\frac{\partial\Psi}{\partial t}+a\,\nabla^2\Psi-V(\mathbf{r})\,\Psi+s\,|\Psi|^2\,\Psi=0, \tag{1.2}$$

where Ψ is the wavefunction, a and s are arbitrary parameters, and $V(\mathbf{r})$ is an external potential function. The parameter s represents the local nonlinear interaction response and its sign determines if the NLSE is attractive ($s > 0$) or repulsive ($s < 0$). The modulus-squared of the wavefunction, $|\Psi|^2$, represents the observable which, in the case of BECs, is the boson density, while in nonlinear optic settings, is the intensity of light. The parameters a and s could be eliminated from Eq. (1.2) through rescaling space and Ψ (leaving s to be either 1 or -1) but they are left as general parameters in order to make it easier to translate results obtained from Eq. (1.2) to those for a particular application.

Due to the oscillatory nature of the real and imaginary parts of Ψ, a subset of solutions to the NLSE (dubbed 'steady-states') have an associated frequency which we denote as Ω in which, starting at time $t = 0$, the real and imaginary parts of Ψ return to their original positions at time $t = 2\,\pi\,n/\Omega, n = 1, 2, 3, ...$ Steady-state solutions are solutions to the NLSE that have a constant modulus $|\Psi|$ over time, in which case it can be represented as $\Psi = U(x, y, z)\exp[i\,\Omega\,t]$, where U is a time-independent function. In many previous studies, the frequency Ω is included in the NLSE itself by using the transformation $\Psi \rightarrow \Psi\exp[i\,\Omega\,t]$, which adds a linear term to Eq. (1.2) of $-\Omega\,\Psi$. However, we have not found this formulation to be of considerable help in our study and have therefore not utilized it. The simplest non-trivial steady-state solution to the NLSE is a 'plane-wave' solution defined as $\Psi = \sqrt{\Psi_\infty}\exp[i\,\Omega\,t]$, where $\sqrt{\Psi_\infty}$ is a real constant. When inserted into Eq. (1.2), the plane-wave solution yields $\Psi_\infty = \Omega/s$, which shows that the sign of Ω must be equal to the sign of s.

Another important quantity in the NLSE is a typical length scale known as the 'healing length'. This is a length scale a_H where the second and fourth terms in the NLSE of Eq. (1.2) are of the same magnitude. In the context of BECs, this equates to where the kinetic energy and the particle interactions are balanced. Taking ∇^2 to be a_H^{-2}, and

$|\Psi|^2 \sim \Psi_\infty = \Omega/s$, we have $|a\Psi/a_H^2| = |s\Psi(\Omega/s)|$ which yields a healing length of

$$a_H = \sqrt{\frac{a}{|\Omega|}}. \tag{1.3}$$

The healing length is an important length scale that is prevalent in the description of the width of localized structures, most notably in the present context as the approximate radius of the cores of vortices (see the next section and Chapter 2 for details).

In this dissertation, we will almost always assume that there is an infinite constant-density background, and thus the external potential term $V(\mathbf{r})$ is set to zero (although it is included in the developed codes). This eases the obtaining of analytical results, and allows for simplified dynamics to be studied (as interactions with the external potential are not present). The knowledge gathered can then be extended into specific applications or more complicated modifications of the NLSE. For example, the study of modulational instability of vortices in the cubic NLSE in Ref. [13] laid the framework for the study in Ref. [14] of the more complicated stability and dynamics of vortices in the cubic-quintic NLSE, which has closer direct relevance to application (in that case, nonlinear optics).

1.1.2 Vortices and Vortex Rings

One of the most interesting family of solutions to the NLSE are those which exhibit topological charge. In two dimensions, these coherent structures correspond to vortices while, in three-dimensions, they can correspond to quantized vortex rings. Dark vortex rings in the repulsive NLSE are ring-shaped structures which, due to their vorticity, form a toroidal area whose center ring has zero-density, with increasing density away from the center which, in the absence of an external potential, converges to a constant-density background. They have an intrinsic transverse velocity due to their vorticity [15], which can be overcome by considering the vortex rings in a co-moving reference frame, or by counteracting the velocity by a trapping potential [16].

Bright vortex ring solutions in the attractive NLSE also exist, in which case the vortex ring resembles a smoke-ring in that it has a well-defined radius of greatest density which trails off to zero towards the inside and outside of the ring. However, as in the case of bright two-dimensional vortices [13], bright vortex rings are unstable and undergo collapse. As such, their study is not as physically relevant as the dark vortex rings, but are worth a brief exploration (especially since such rings realized in the cubic-quintic NLSE (CQNLS), may be stable for some parameter ranges as is the case with two-dimensional bright vortices in the CQNLS [14]).

Two-dimensional dark vortices have been observed in many experiments in quantum systems including BECs and nonlinear optics (see Ref. [17] for an extensive list of studies). Dark vortex rings have been seen experimentally in super-fluid helium [18, 19] as well as in the context of BECs in the decay of dark solitons in two-component BECs [20], direct density engineering [21–23], and in the evolution of colliding symmetric defects [24]. Experimental transmission images of vortex rings in a repulsive BEC are shown in Fig. 1.1.

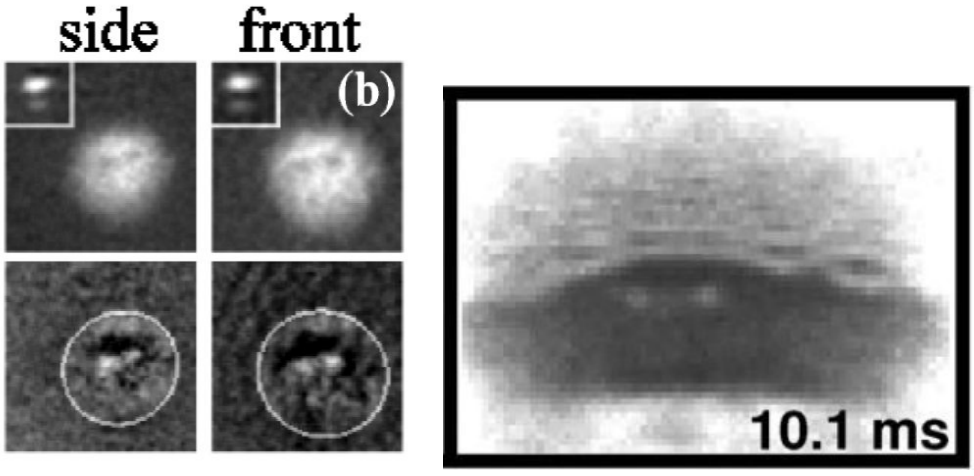


Figure 1.1. Experimental images of a vortex ring in a BEC. The left image is reproduced with permission from Fig. 4 of Ref. [20], while the right image is reproduced with permission from Fig. 2(a) of Ref. [24].

Numerical studies of experimentally feasible dark vortex-ring generation in BECs have also been explored in the context of collisions of multiple BECs [25, 26], flows past an obstacle [27], Bloch oscillations in an optical trap [28], collapse of bubbles [29], instability of rarefaction two-dimensional pulses [30], flow past a positive ion [31, 32] or an electron bubble [33], crow instability of two vortex lines [34], and dragging of an object through a BEC [35].

The energy and transverse velocity of a single dark vortex ring in the NLSE has been studied analytically [15, 36, 37]. The velocity results have been numerically verified in more recent studies [38], although the details of the comparisons were not shown. In Ref. [39], the authors derived and numerically simulated a continuous family of axisymmetric co-moving structures in the NLSE in which two types are found: rarefaction solitary pulses with no vorticity and vortex rings. In Ref. [40], collisions between the rarefaction pulses and vortex lines and rings were studied.

Numerical studies directly focused on the structure and stability of dark vortex rings in trapped BECs have also been done. These include the generation of steady-state vortex rings in toroidal [41] and harmonic [16, 42] trapping potentials, multiple parallel ring structures [43], as well as the study of the bending-wave instability of vortex rings [44] and their degeneration into quantum turbulence [45]. Some dynamics of vortex rings in trapped BECs have been explored, such as their oscillatory motion resulting in self-annihilation [46], and collisions of vortex rings with solitons in cylindrical BECs [47, 48].

Collisions and interactions between vortex rings in a NLSE setting (such as an infinite-extent homogeneous BEC with no trapping potential) is of main interest in the current study. A previous study of the collisions of two dark vortex rings was performed in Ref. [38], but for only a few configurations, and without quantitative analysis. An extensive dynamical study was not done most likely due to the large computational cost of the three-dimensional simulations at the time even when using a large super computer

(typically each simulation took a few hours to complete [49]). The computational techniques developed and utilized in this dissertation eliminate that obstacle and allow for a more thorough study of the interacting rings. Studies on the interaction and collisions of numerous vortex rings of various radii in random locations has been done in order to study the quantum turbulence that arises out of the numerous collisions [50], but controlled interactions of multiple rings has not been presented.

1.1.3 High-Order Compact Schemes

For almost every nonlinear partial differential equation (PDE), including the NLSE, the most interesting solutions require simulations using numerical methods. There exists many numerical methods for integrating the NLSE (a review of the most common methods can be found in Ref. [51]). A common methodology is to discretize the solution in space by using finite-difference approximation schemes for the required derivatives. The resulting semi-discrete equation is then integrated in time using a finite-difference time-stepping scheme which can be independent of the spatial scheme, combined with the spatial scheme, explicit (the next step at each grid point is purely a function of the domain at the previous step(s)) or implicit (the next step at each grid point is the solution to a linear system involving the grid points at the previous step(s) as well as other grid points at the next step).

A *high-order* spatial finite-difference scheme is one which exhibits greater than second-order ($O(h^2)$, where h is the grid spacing) accuracy. While there are a large number of scenarios where higher-order schemes are a necessity due to the desired accuracy of the simulations, often the increased accuracy of high-order schemes is unnecessary, and second-order accuracy is sufficient for the problem. However, even in such a case, using high-order schemes is still desirable because they allow one to simulate a solution with many fewer grid points while maintaining the same accuracy as a second-order scheme. In those situations, the desired grid point size is based on the ability to resolve the structure of the solution, and not on the accuracy of computation.

Higher-order finite-difference schemes are typically achieved by computing derivatives with a wider scheme stencil. This adds a difficulty near the boundary, as one must be able to calculate the inner point near the boundary with the same accuracy as the internal scheme which can be complicated to implement. If this is not done, the entire simulation can eventually become lower-order. Another disadvantage of wide-stencils is that they cause many parallel implementations to be more difficult to realize, as different compute nodes must share or transfer more boundary values to their neighboring nodes. Besides the added complexity of the codes, the extra communication (in the case of a CPU cluster) and/or global-memory-access (in the case of graphical processing units) reduces the overall parallel performance [52].

For the aforementioned reasons, there is great interest in high-order compact (HOC) schemes. These are finite-difference schemes which exhibit higher-order accuracy but still only rely on the closest neighboring points for computations. HOC schemes have been developed for various multidimensional steady-state PDEs including the convection-diffusion equation [53], Poisson's equation [54], the stream-function vorticity form of the Navier-Stokes equations [55], as well as generalized linear elliptical PDEs with variable coefficients [56] to name a few. Many HOC implementations of time-dependent PDEs have also been formulated including Burger's equation [57], the wave equation [58], the Euler equations [59] and the time-dependent convection-diffusion equation [60, 61] as well as others. Time-dependent HOC schemes have also been developed for the one- and two-dimensional linear [62, 63] and one-dimensional nonlinear Schrödinger equations [64, 65]. One drawback of HOC schemes are that their formulations are typically specific for the model equation being used, and require re-deriving the schemes for each different PDE to be simulated (see Ref. [56], where the authors developed a Maple program to symbolically generate a HOC scheme for general three-dimensional linear elliptical PDEs).

The time-dependent HOC schemes developed so far are all implicit (even if an otherwise explicit time-stepping scheme is inserted into the formulation of the HOC scheme), requiring solving a linear system in each time-step, as well as iterative techniques when simulating nonlinear PDEs such as the NLSE (see however Ref. [64] where the author's second implicit scheme implementation was able to be formulated to avoid the need of nonlinear iterations). The computational and storage requirements for implicit schemes can be prohibitive in large multi-dimensional settings. They are also, in general, difficult to optimally parallelize, especially in higher dimensions. It is therefore desirable to find a way to have a compact scheme which remains fully explicit. In Chapter 3, such a scheme is developed for use with the NLSE.

1.1.4 Numerical Boundary Conditions

When utilizing numerical methods to approximate the solutions to time-dependent PDEs, including the NLSE, proper handling of boundary conditions can be quite challenging. Sometimes, an otherwise stable numerical scheme will become unstable depending on how the boundary conditions are computed [66]. In addition, high-order schemes can degrade in accuracy to lower-order when using boundary conditions which are not compatible with the high-order accuracy [67]. Proper handling of boundary conditions in higher-order schemes, especially in high-order compact schemes can be even more of a challenge [68, 69].

Many problems are defined on a limited domain with specified boundary conditions. The most common are Dirichlet (where the values of the function are specified at the boundaries), Neumann (where the values of the spatial derivatives of the function are specified at the boundaries), Robin (where a linear combination of Dirichlet and Neumann conditions are used on the boundaries), Cauchy (where both Dirichlet and Neumann conditions are simultaneously applied on the boundaries), and mixed (where variant boundary conditions are applied to different boundary regions of the numerical domain). In all such cases, the numerical implementation of the boundary conditions can

typically be formulated based on their definition (such as using finite difference approximations for spatial derivatives).

A more complicated situation exists for problems which reside in an infinite domain in which case the need for boundary conditions is only due to the finite-sized numerical domain. In such open boundary problems, one often ideally wants a transparent boundary condition. One approach is to use a form of the Sommerfeld radiation condition at the boundaries utilizing interior grid point values to approximate the outward phase velocity [70].

In the context of the linear and nonlinear Schrödinger equations, many sophisticated boundary conditions have been developed which simulate transparent or artificial boundaries [71]. For the linear Schrödinger equation, these include continuous, pole condition, temporally discrete, spatially discrete, fully discrete, as well as others in one dimension, while in two dimensions, continuous transparent boundary conditions have been formulated which make use of the Sommerfeld radiation condition. Work has also been done in formulating similar conditions for the one-dimensional NLSE (see Ref. [71] and references therein for a complete review).

Most of the aforementioned boundary conditions focus on eliminating reflections off the boundaries when studying dynamics of solutions which tend to zero at infinity. While these boundary conditions would be valid for the study of bright structures in the NLSE (such as the bright solitons and bright vortices described in Chapter 2), since the main focus of this dissertation is the study of dark vortex rings which exist in a constant-density background, we do not make use of the boundary conditions described therein. Also, the boundary conditions shown in Ref. [71] can be very complicated to implement, and are therefore best suited for large projects. A relevant exception to this is the Sommerfeld radiation condition described in Ref. [70], which is relatively easy to implement and has been previously successful in simulating a constant-density background in the NLSE [32]. However, since it requires additional computations at the

boundaries, and there has been some discussion debating its usefulness in some cases (see Ref. [72]), an alternative approach to approximate a constant-density background could be useful.

Often, researchers will forgo a complicated boundary condition implementation and instead use tried-and-true boundary condition techniques which are very simple yet provide acceptable results. One of the most common is the use of Dirichlet boundary conditions when simulating solutions which decay towards zero at infinity, and where most of the dynamics (or 'action') is expected to remain in the central regions of the computational grid (since the main issue with using Dirichlet boundary conditions on an infinite domain problem is reflections off the boundary). However, infinite-domain problems involving PDEs whose function values are complex (such as the NLSE) cannot, in general, make use of standard Dirichlet boundary conditions because of the constant oscillation of the real and imaginary parts of the function due to the intrinsic frequency of the system and the dynamics of the solutions (when both the real and imaginary parts of the solution decay to zero at infinity, standard Dirichlet conditions of zero-value *can* be used). There are however, many solutions such as dark vortex rings in the NLSE where the modulus-squared of the solution converges to a constant value at infinity. In such a case, a modulus-squared Dirichlet (MSD) boundary condition which keeps the modulus-squared of the solution at the boundaries constant would be desirable. In Chapter 4, a new MSD boundary condition is formulated for general complex time-dependent PDEs and is tested for use with the NLSE.

1.1.5 Code Parallelism with Graphics Processing Units

Many interesting problems (especially those in high dimensions) require very large simulations. In such cases, it is desirable to use code run on parallel computational systems that can greatly reduce the simulation time. Generally, this means running codes on expensive large computer clusters through the MPI and/or OpenMP application

programming interfaces (APIs). However, recently, a new breed of parallel computing has been gaining great interest, that of general purpose graphical processing units (GPUs).

Graphical processing units were first developed in order to allow graphics cards to parallelize their massive computations to speed up video games and image processing. It was then realized that such hardware could be adapted to run scientific codes as well. This led to the development of code APIs which allows one to write code which takes advantage of the GPU hardware. NVIDIA cooperation is the leading company in this new form of computing, developing the compute unified device architecture (CUDA) API which can be compiled to run scientific codes on NVIDIA's line of GPU video cards, as well as non-video stand-alone GPUs designed specifically for parallel computation.

For many problems, GPU computing is seen as a large improvement over previous parallel techniques since access to a large cluster can be expensive or not available. A typical GPU (as of this writing) can have up to 1024 processing cores and up to 6GB of RAM, with a throughput of over a Tera-FLOP on a single card. The price of the GPUs is another major factor, as one (as of this writing) can purchase an off-the-shelve GPU with 512 cores, and 1.5GB of RAM for under $420. This allows for super-computing capabilities to be realized on a single desktop PC for medium-sized problems.

GPUs have been used for various finite-difference scheme codes with good results [73–78]. In Ref. [73], the authors wrote code to simulate the two-dimensional Maxwell equations yielding a speedup of up to ten times when compared to the CPUs of the day (2004). They did this before CUDA was developed using assembly code directly. Ref. [74] also did not utilize CUDA for their simulations of the three-dimensional Maxwell equations, yielding speedups of over 100 times. The two-dimensional Maxwell equations were simulated in Ref. [75] using CUDA code where speedups of 50 were reported. In Ref. [76], the authors simulate the three-dimensional wave equation with an eighth-order finite-difference scheme using CUDA on multiple GPUs. A similar multi-GPU code was shown in Ref. [77] for the three-dimensional wave equation using a

fourth-order finite-difference scheme where speedups of 60 were reported. In Ref. [78], the authors use CUDA to simulate the multi-dimensional complex Ginzburg-Landau equation (a generalization of the NLSE), but speedups versus CPU codes were not reported as their main goal was to compare the CUDA codes using a complex number C structure versus using explicit treatment of the real and imaginary parts of the solution. All these studies indicate that a GPU treatment of the NLSE would yield significant performance improvements. In Chapter 7, such GPU-based integrators for the NLSE are developed and tested with positive results.

1.1.6 Reusable Software Package

In many cases of numerical research, researchers will produce codes to accomplish their simulations and, after the project is completed, the codes are never used again (or by anyone else). Thus, other researchers in the field must produce their own codes, a classic case of re-inventing the wheel. Although every problem is different and demands different techniques, producing a freely distributed code package that is powerful and easy to use can help a wide range of researchers in the field, especially since often codes are easily altered to conform to the specific problem and model equations of the user.

MATLAB is a mathematical computational software program created by MathWorks Inc. It has become a standard in many academic institutions as well as in industry for mathematical computations. It is an easy-to-use platform for writing and running mathematical codes including built-in analytics, visualization, and other useful functions. MATLAB would therefore seem to be an ideal platform for developing reusable codes.

However, there is a prevalent opinion that while MATLAB may be useful for course work, small-scale codes, and learning purposes, for serious calculations it is simply too slow to be useful. This is because the typical way to realize a MATLAB code is to write script files which contain MATLAB commands and code which is run line-by-line and not compiled (as MATLAB code is a scripting language). Although MATLAB has

built-in vectorized computations, the script codes still run very slow compared to compiled C or FORTRAN codes.

There *is* a way to achieve efficient large-scale codes in MATLAB which is is to write custom C codes which connect to MATLAB through an interface compiler called MEX. Once the C codes have been written to be MEX-compatible and compiled, they can be called from a MATLAB script. The codes run as fast as an equivalent stand-alone C code, with minimal overhead, and are up to 10 times faster than the equivalent script codes (see Chapter 6). The C codes written for the MEX interface can be written to only have custom MATLAB-based code for the input of data from MATLAB and the returning of the computation results back to MATLAB. Therefore, the core of the C code is standard C which allows it to be portable in case the MATLAB interface is no longer desired.

Since efficient large-scale codes can be realized in the MATLAB environment, for the aforementioned reasons, this makes MATLAB an ideal platform for redistributable code packages. In order for the codes to be visible and available to researchers all over the world, it is a very useful tool to have the codes on a dedicated website. In Chapter 8, we describe the distributable code package developed in this dissertation named NLSEmagic.

1.2 OUTLINE OF THE STUDY

The current study encompasses all levels of a typical computational science problem — mathematical models, numerical algorithms, code development, and numerical simulations. Although some of the topics discussed are, by nature, somewhat independent from each other, we have tried to link the chapters together as much as possible. The dissertation is outlined as follows: In Chapter 2, an overview of solitons, vortices, and vortex rings is presented to set the stage for the numerical study of the dynamics of vortex rings. The basic mathematical framework and analytical results are given. Chapter 3 discusses the numerical algorithms used to integrate the NLSE focusing on a two-step high-order compact scheme designed to be useful when using explicit schemes for time-dependent problems. Then, in Chapter 4 we describe a new

modulus-squared boundary condition for complex-valued PDEs and its application to the simulation of dark structures in the NLSE (such as vortex rings) amidst an infinite-extent constant-density background. We wrap up the discussion of numerical methods in Chapter 5 by performing a detailed stability analysis of the numerical schemes. The stability results obtained are vital for performing efficient simulations using the schemes. The code implementation of the numerical schemes applied to the NLSE is discussed in the following three chapters. Chapter 6 describes the MATLAB script and MEX integrator codes, while Chapter 7 describes in detail the CUDA code implementation of the integrators for use on GPUs. The description and distribution information of the complete code package (called 'NLSEmagic') is summarized in Chapter 8. With the code in hand, the numerical study of the vortex rings in the NLSE begins in Chapter 9 with the structure and dynamics of a single vortex ring. Both dark and bright vortex rings of various charges are formulated, and the dynamics of charge 1 dark vortex rings are studied including their intrinsic transverse velocity and quadrupole oscillations. In Chapter 10 the dynamics of the interaction and collisions of multiple vortex rings is studied including the scattering of off-set co-planar dark vortex ring collisions, and the interactions of vortex rings traveling together. Choreographed multi-vortex ring scenarios are also presented. The results of each chapter are then summarized in a conclusion given in Chapter 11. Publications arising out of the study as well as suggestions for future extensions are discussed as well.

CHAPTER 2

OVERVIEW OF SOLITONS, VORTICES, AND VORTEX RINGS

In this chapter, we begin our study of vortex rings in the NLSE by exploring the relevant analytical results describing their structure and dynamics for use with the numerical studies of Chapters 9 and 10. Since vortex rings are topological extensions of two-dimensional vortices, whose radial profiles resemble one-dimensional solitons, it is also important to briefly review the basic structure and dynamics of the lower-dimensional structures before discussing the vortex rings themselves. Throughout the dissertation, the lower-dimensional structures described in this chapter will also be used to test numerical schemes (Chapter 3), boundary conditions (Chapter 4), and GPU code speedups (Chapter 7).

As mentioned in Sec. 1.1.1, the sign of the parameter s in the NLSE of Eq (1.2) determines whether the NLSE is attractive ($s > 0$) or repulsive ($s < 0$). Coherent structures exist in each case but are quite distinct. In the attractive case, they are characterized by a localized density ($|\Psi|^2$) structure amidst a zero-background (and are therefore called 'bright' structures), which in higher dimensions are typically unstable. In the repulsive case, the structures are characterized as an area of little or no density amidst a constant or locally high level density background (and therefore referred to as 'dark' structures). The dark vortex rings which are the main focus of this dissertation fall into this category.

2.1 ONE-DIMENSIONAL SOLITONS

A soliton can be defined as a self-reinforcing wave packet which moves at a constant velocity (which may be zero) due to the dispersive and nonlinear effects of the medium canceling each other out [79].

The one-dimensional steady-state bright soliton solution to the NLSE of Eq. (1.2) with $V(x) = 0$ centered about $x = 0$ is given by [80]

$$\Psi(x,t) = \sqrt{\frac{2\Omega}{s}}\,\text{sech}\left[\sqrt{\frac{\Omega}{a}}\,x\right]\,\exp\left[i\,\Omega\,t\right], \tag{2.1}$$

where a and s are parameters of the NLSE of Eq (1.2), and Ω is the frequency. The soliton of Eq. (2.1) can be made to move at a constant velocity c by situating it amidst a co-moving back-flow (for details, see Appendix B). The resulting traveling soliton is given by

$$\Psi(x,t) = \sqrt{\frac{2\Omega}{s}}\,\text{sech}\left[\sqrt{\frac{\Omega}{a}}\,(x - c\,t)\right]\,\exp\left(i\left[\frac{c}{2a}\,x + \left(\Omega - \frac{c^2}{4a}\right)t\right]\right). \tag{2.2}$$

In Fig. 2.1, we show an example of the bright solitons of Eqs. (2.1) and (2.2).

In the repulsive NLSE, a moving solitary solution to the NLSE with $V(x) = 0$ centered about $x = 0$ when $t = 0$ is given by [80]

$$\Psi(x,t) = \sqrt{\frac{\Omega}{s}}\left(i\,\frac{c}{v} + \sqrt{1 - \frac{c^2}{v^2}}\,\tanh\left[\sqrt{\frac{\Omega}{2a}}(x - c\,t)\,\sqrt{1 - \frac{c^2}{v^2}}\right]\right)\exp\left[i\,\Omega\,t\right], \tag{2.3}$$

where $v = \sqrt{-2a\Omega}$ is called the sound velocity, c is the velocity of the soliton, and $\Omega < 0$ is once again the frequency. Equation (2.3) is refereed to as a 'gray' soliton as it describes a moving self-sustaining drop in density, but which does not reach zero-density at its center (unless $c = 0$). The value of the density at the center of the wave packet decreases with lower values of the velocity c. In the case where $c = 0$, the density at the center of the soliton reaches zero and the resulting steady-state solution is referred to as a 'dark' soliton

and is given by

$$\Psi(x,t) = \sqrt{\frac{\Omega}{s}} \tanh\left[\sqrt{\frac{\Omega}{2a}}\, x\right] \exp\left[i\,\Omega\, t\right]. \tag{2.4}$$

The dark soliton of Eq (2.4) can be modified to move at a constant velocity while maintaining its profile by embedding it in a co-moving back-flow in the same manner as the bright soliton of Eq. (2.2) yielding

$$\Psi(x,t) = \sqrt{\frac{\Omega}{s}} \tanh\left[\sqrt{\frac{\Omega}{2a}}\,(x - c\,t)\right] \exp\left(i\left[\frac{c}{2a}\,x + \left(\Omega - \frac{c^2}{4a}\right) t\right]\right). \tag{2.5}$$

In Fig. 2.2 we show examples of gray and dark solitons.

2.2 VORTICITY

Before discussing coherent structures in higher dimensions, it is useful to first introduce the quantity called vorticity. Vorticity is a measure of the change in rotational fluid velocity. As the NLSE can be written in a fluid-dynamical form [81], vorticity is a relevant quantity in the context of the NLSE, and it is a prime indicator of the presence of vortices.

The vorticity in the three-dimensional NLSE is a vector quantity defined by

$$\vec{\omega} = \nabla \times \nabla P, \tag{2.6}$$

where P is the phase of the solution Ψ defined as

$$P(x,y,z,t) = \arg(\Psi(x,y,z,t)). \tag{2.7}$$

Because certain solutions such as vortices create a discontinuity in P, the quantity ∇P in Eq. (2.6) is typically expressed as

$$\nabla P = \frac{i}{2}\frac{\Psi\nabla\Psi^* - \Psi^*\nabla\Psi}{|\Psi|^2}. \tag{2.8}$$

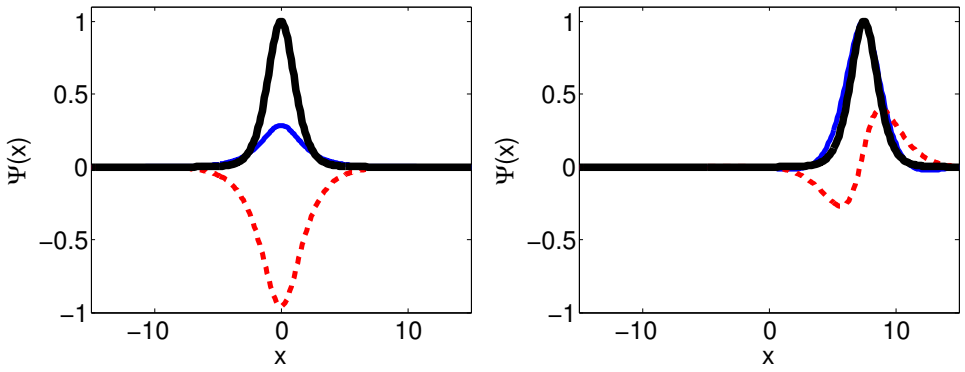


Figure 2.1. Depiction of steady-state (left) and moving (right) one-dimensional bright solitons of the one-dimensional NLSE. The thick (black) line represents the density ($|\Psi|^2$) of the solution, while the thin (blue) and dashed (red) lines represent the real and imaginary parts of the wavefunction Ψ respectively. In all cases the soliton is shown at $t = 10$ with parameters $a = 1$, $s = 1$, and $\Omega = 0.5$. The moving soliton has a velocity of $c = 0.75$.

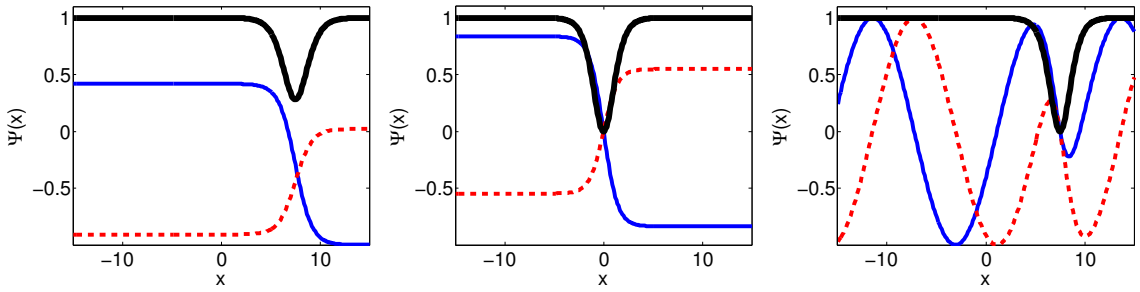


Figure 2.2. Top: Depiction of a gray soliton in the one-dimensional NLSE. Bottom: Depiction of a steady-state (left) and a co-moving (right) dark soliton solution. Both the gray and co-moving dark solitons have a velocity of with $c = 0.75$. In all cases the soliton is shown at $t = 10$ with parameters $a = 1$, $s = -1$, and $\Omega = -1$. Additional figure descriptions are the same as those in Fig. 2.1.

Using Eq. (2.8), the vorticity $\vec{\omega}$ of Eq. (2.6) can be written out as

$$[\omega_x, \omega_y, \omega_z] = \left[\left(\frac{\partial \mathcal{V}_z}{\partial y} - \frac{\partial \mathcal{V}_y}{\partial z}\right), \left(\frac{\partial \mathcal{V}_x}{\partial z} - \frac{\partial \mathcal{V}_z}{\partial x}\right), \left(\frac{\partial \mathcal{V}_y}{\partial x} - \frac{\partial \mathcal{V}_x}{\partial y}\right)\right], \tag{2.9}$$

where $\vec{\mathcal{V}}$ is defined as

$$[\mathcal{V}_x, \mathcal{V}_y, \mathcal{V}_z] = \nabla P = \frac{i}{2}\frac{\Psi\nabla\Psi^* - \Psi^*\nabla\Psi}{|\Psi|^2}, \tag{2.10}$$

where

$$\nabla\Psi = \left[\frac{\partial\Psi}{\partial x}, \frac{\partial\Psi}{\partial y}, \frac{\partial\Psi}{\partial z}\right]. \tag{2.11}$$

Although the vorticity is a vector quantity, the L2-norm of $\vec{\omega}$ is often used as a scalar indication of vorticity.

In two-dimensional settings, the vorticity becomes a scaler quantity defined as

$$\omega = \frac{\partial \mathcal{V}_y}{\partial x} - \frac{\partial \mathcal{V}_x}{\partial y}, \tag{2.12}$$

where

$$[\mathcal{V}_x, \mathcal{V}_y] = \nabla P(x, y, t) = \frac{i}{2}\frac{\Psi\nabla\Psi^* - \Psi^*\nabla\Psi}{|\Psi|^2}, \tag{2.13}$$

and

$$\nabla\Psi = \left[\frac{\partial\Psi}{\partial x}, \frac{\partial\Psi}{\partial y}\right]. \tag{2.14}$$

Since the norm of the vorticity is a very high peak at the location of a vortex and drops to zero rapidly away from the vortex core, it is an ideal quantity to be able to numerically track the position of a vortex or vortex ring whether or not they are dark or bright (for details, see Chapter 9). One problem with the numerical computation of the vorticity is that if $|\Psi|^2 = 0$ anywhere in the solution (such at the center of vortices), then the form of ∇P of Eq. (2.8) suffers from a singularity. Therefore, when computing the vorticity during a numerical simulation, a small term $0 < \epsilon_v \ll 1$ is added to the denominator of

Eq. (2.8) yielding

$$\nabla P \approx \frac{i}{2}\frac{\Psi\nabla\Psi^* - \Psi^*\nabla\Psi}{|\Psi|^2 + \epsilon_v}. \tag{2.15}$$

The difference between the true vorticity and that using Eq. (2.15) is typically minimal away from singularities, and since the actual quantity of the vorticity is not usually sought after, but rather the relative intensity of vorticity, for our uses, adding ϵ_v does not have a noticeable impact.

2.3 TWO-DIMENSIONAL VORTICES

A vortex in the NLSE is defined as a coherent structure which exhibits a topological defect whose circulation is quantized [82]. The number of complete phase rotations in the azimuthal direction of the vortex is refereed to as the vortex's 'charge', denoted as m. In this section we describe the most basic properties of the structure and dynamics of bright and dark vortices.

2.3.1 General Steady-State Form and ODE Profile

The general steady-state solution for both bright and dark vortices in polar coordinates centered around $r = \sqrt{(x-x_0)^2 + (y-y_0)^2}$ with $V(\mathbf{r}) = 0$ is given as

$$\Psi(r,\theta,t) = f(r)\exp\left[i\left(m\,\theta + \Omega\,t\right)\right], \tag{2.16}$$

where $f(r)$ is a real-valued radial profile, m is the vortex charge, and Ω is the frequency. Equation (2.16) can be inserted into the two-dimensional polar NLSE with $V(\mathbf{r}) = 0$ given by

$$i\frac{\partial\Psi}{\partial t} + a\left[\frac{1}{r}\frac{\partial\Psi}{\partial r} + \frac{\partial^2\Psi}{\partial r^2} + \frac{1}{r^2}\frac{\partial^2\Psi}{\partial\theta^2}\right] + s\,|\Psi|^2\,\Psi = 0$$

to yield a one-dimensional ODE for the radial profile $f(r)$ given by

$$-\left(\Omega + \frac{a\,m^2}{r^2}\right)f(r) + a\left(\frac{1}{r}\frac{df}{dr} + \frac{d^2f}{dr^2}\right) + s\,f^3(r) = 0. \tag{2.17}$$

There is no closed-form solution for $f(r)$ for either bright or dark vortices. Therefore, $f(r)$ must be found numerically (see Chapter 9 for details), although as we will show in the following two sections, very good analytical asymptotic estimates of $f(r)$ can be found. We note that the ODE of Eq. (2.17) is also the governing equation for the radial profile of a reticular vortex line in the three-dimensional NLSE.

2.3.2 Structure and Dynamics of Bright Vortices

The density structure of a bright vortex resembles a bright soliton which has been drawn around into a circle to form a ring of density. Using a variational approach with asymptotic assumptions, an extremely close analytical approximation to the vortex radial profile is found to be [13]

$$f(r) \approx \sqrt{\frac{3\,\Omega}{s}}\,\mathrm{sech}\left[\sqrt{\frac{3\,\Omega}{2\,a}}\left(r-\sqrt{\frac{2\,a\,m^2}{\Omega}}\right)\right]. \tag{2.18}$$

The profile of Eq. (2.18) is more accurate for larger values of the charge m, and is less accurate over smaller values of r for low charge vortices (see Ref. [13] for details). However, even for lower charge vortices, the profile of Eq. (2.18) very accurately approximates the radius and peak value of the vortex profile. The radius of the bright vortex is proportional to the magnitude of the charge ($|m|$) and is approximated as [13]

$$r_c \approx \sqrt{2}\,|m|\sqrt{\frac{a}{\Omega}}, \tag{2.19}$$

where we note that $\sqrt{a/\Omega}$ is the healing length discussed in Sec. 1.1.1. The maximum amplitude of $f(r)$ is approximated as

$$A \approx \sqrt{\frac{3\Omega}{s}}. \tag{2.20}$$

One of the advantages of having an accurate closed-form approximation to $f(r)$ is that it can be used as an initial condition for numerical optimization of the ODE of Eq. (2.17) in

order to find numerically exact vortex profiles (see Chapter 9). In Fig. 2.3, we display some examples of bright vortices in the NLSE. We note that the ring-shaped vorticity of the charge $m = 3$ vortex is a numerical artifact caused by the artificial ($\epsilon_v = 0.001$) term in the vorticity calculations described in Sec. 2.2, which in this case, has a noticeable effect on the computed vorticity.

Bright vortices are known to be azimuthally modulationally unstable. It can be shown (see Ref. [13]) that the maximum growth rate of the most unstable mode is approximately

$$\lambda_{max} \approx 2\,\Omega, \tag{2.21}$$

while the strongest growing azimuthal mode is approximately

$$K_{max} \approx 2|m|, \tag{2.22}$$

which is therefore the approximate number of filaments that the vortex will break up into. The critical mode, above which all modes are stable is approximated as

$$K_{crit} \approx 2\sqrt{2}|m|. \tag{2.23}$$

An example of azimuthal break up is shown in Fig. 2.4. It is worth mentioning here that the corresponding vortices within the cubic-quintic NLSE can be azimuthally *stable* depending on the value of Ω (see Ref. [14] for details).

2.3.3 Structure and Dynamics of Dark Vortices

Dark vortex solutions to the repulsive NLSE with $V(\mathbf{r}) = 0$ have a core density of zero (similar to the bright vortices) which increases radially. The main difference in the density structure in the case of a dark vortex is that the density grows until it converges to a background density equivalent to that of the plane-wave solution described in Chapter 1

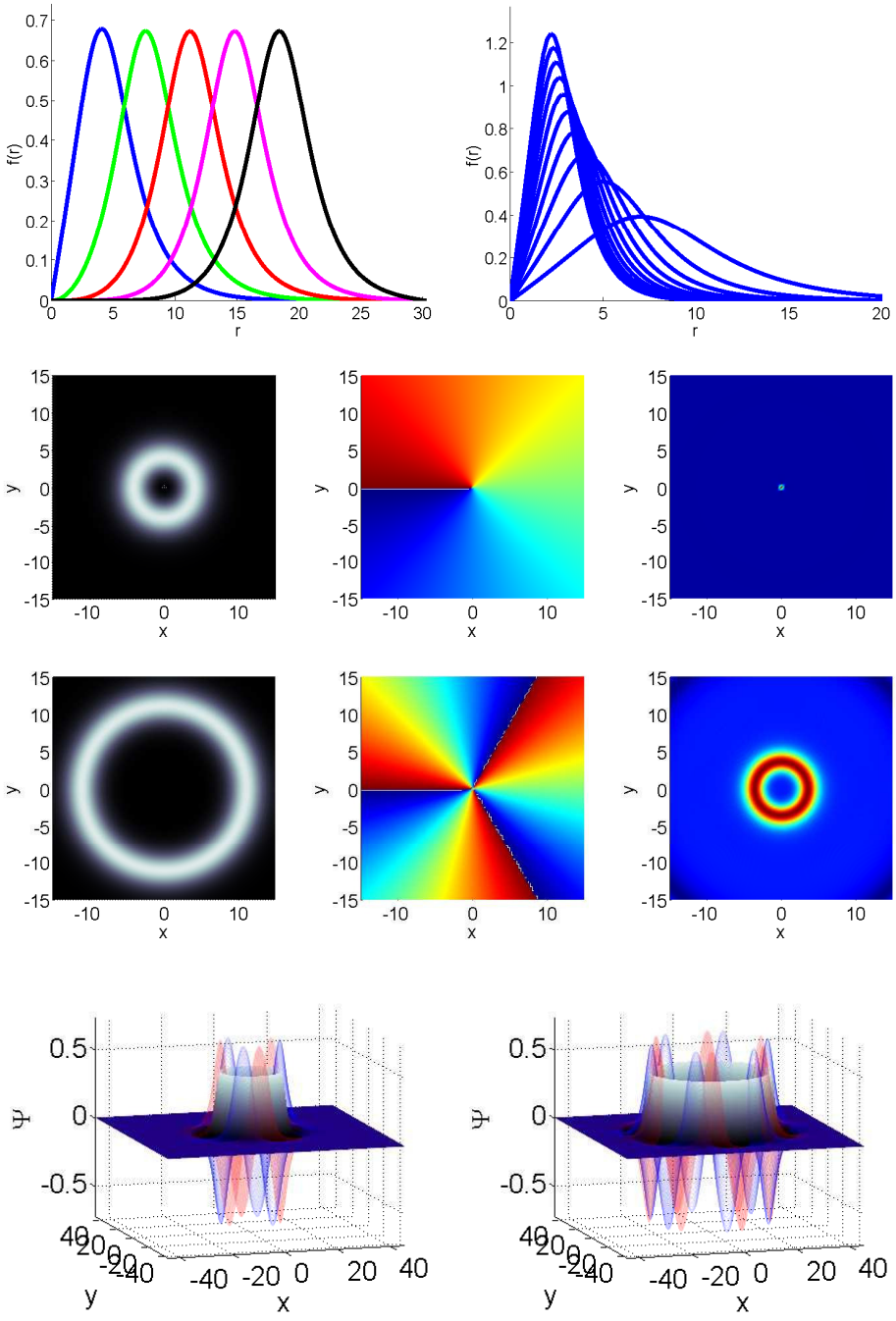


Figure 2.3. Depiction of bright vortices in the two-dimensional NLSE. Parameters for all plots are $a = 1$ and $s = 1$. Top left: Vortex profiles with $\Omega = 0.15$ and charges $m = 1 \to 5$ (left to right). Top right: Vortex profiles with charge $m = 1$ and frequencies $\Omega = 0.05 \to 0.5$ (bottom to top). The two middle rows show the density, phase, and vorticity (left to right) of a $m = 1$ (top) and $m = 3$ (bottom) vortex with $\Omega = 0.15$. The bottom row shows an $m = 3$ and $m = 6$ vortex with $\Omega = 0.15$. The blue and red mesh represents the real and imaginary parts of the wave-function respectively.

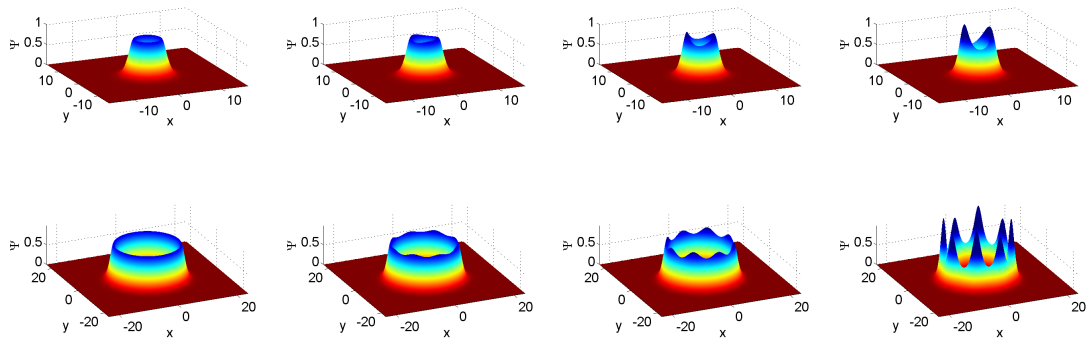


Figure 2.4. Depiction of azimuthal break up of bright vortices in the two-dimensional NLSE. The height of the vortex represents the modulus-squared of Ψ. The top row is a charge $m = 1$ vortex while the bottom row's vortex has a charge of $m = 4$. In each case, a uniformly distributed random perturbation of maximum amplitude $\epsilon = 0.02$ is added to the initial condition and the vortices are simulated to an end time of $t \approx 15$.

given by

$$|\Psi(r \to \infty)|^2 = \Psi_\infty = \frac{\Omega}{s}. \tag{2.24}$$

Therefore, the phase defect of dark vortices are globally 'seen' by each other which allows for interesting dynamics of multiple dark vortices.

As was the case for bright vortices, the radial profile of a dark vortex must be solved for numerically. However by noting that $d\Psi/dr \to 0$ as $r \to \infty$, an analytic asymptotic profile which is valid for large r can be found from Eq. (2.17) as [83]

$$f(r) \approx \text{Re}\left[\sqrt{\frac{\Omega}{s} + \frac{a\,m^2}{s\,r^2}}\right]. \tag{2.25}$$

Although the profile of Eq. (2.25) is only valid for large r, it is useful both as an initial condition to optimization routines, as well as a way to compute the boundary values of a domain containing the vortex (see Chapter 9 for a comparison of the approximate profile to the numerically exact profile). The location where Eq. (2.25) equals zero can be taken as an indicator of the vortex core-radius. This radius is (like the radius of the bright

vortices) proportional to $|m|$ and (remembering that $\Omega < 0$) is approximately

$$r_c \approx |m| \sqrt{-\frac{a}{\Omega}}, \tag{2.26}$$

which is the magnitude of the vortex charge times the healing length. In Fig. 2.5, we show some examples of dark vortices.

Unlike bright vortices, dark vortices of charge $|m| = 1$ are stable. This allows for interesting long-term dynamics of multiple vortices interacting with each other. Even higher charge dark vortices are only weakly unstable, and so can also be studied for long periods of time. The most basic interactions are those of two vortices with either the same or opposite charges. Two equal charge vortices will rotate about each other in a circular path, while two opposite charge vortices will travel in a straight line trajectory perpendicular to their mutual line of sight. For $\Omega = -1$ and $s = -1$, a good estimate of the angular velocity of rotation of two equal charge vortices separated by a distance $2d$ is [84]

$$c_a \approx \frac{a\, m}{d^2}, \tag{2.27}$$

while opposite charged vortices will travel with a linear velocity of approximately

$$c_l \approx \frac{a\, m}{d}. \tag{2.28}$$

If the distance between vortices is large compared to the vortex width, the interaction dynamics of multiple vortices can be described by a simple set of coupled ODEs defined as (for $\Omega = -1$ and $s = -1$) [84]

$$\begin{aligned} \frac{dx_i}{dt} &= -b\, a \sum_{k \neq i} m_k \, \frac{y_i - y_k}{r_{ik}^2}, \\ \frac{dy_i}{dt} &= +b\, a \sum_{k \neq i} m_k \, \frac{x_i - x_k}{r_{ik}^2}, \end{aligned} \tag{2.29}$$

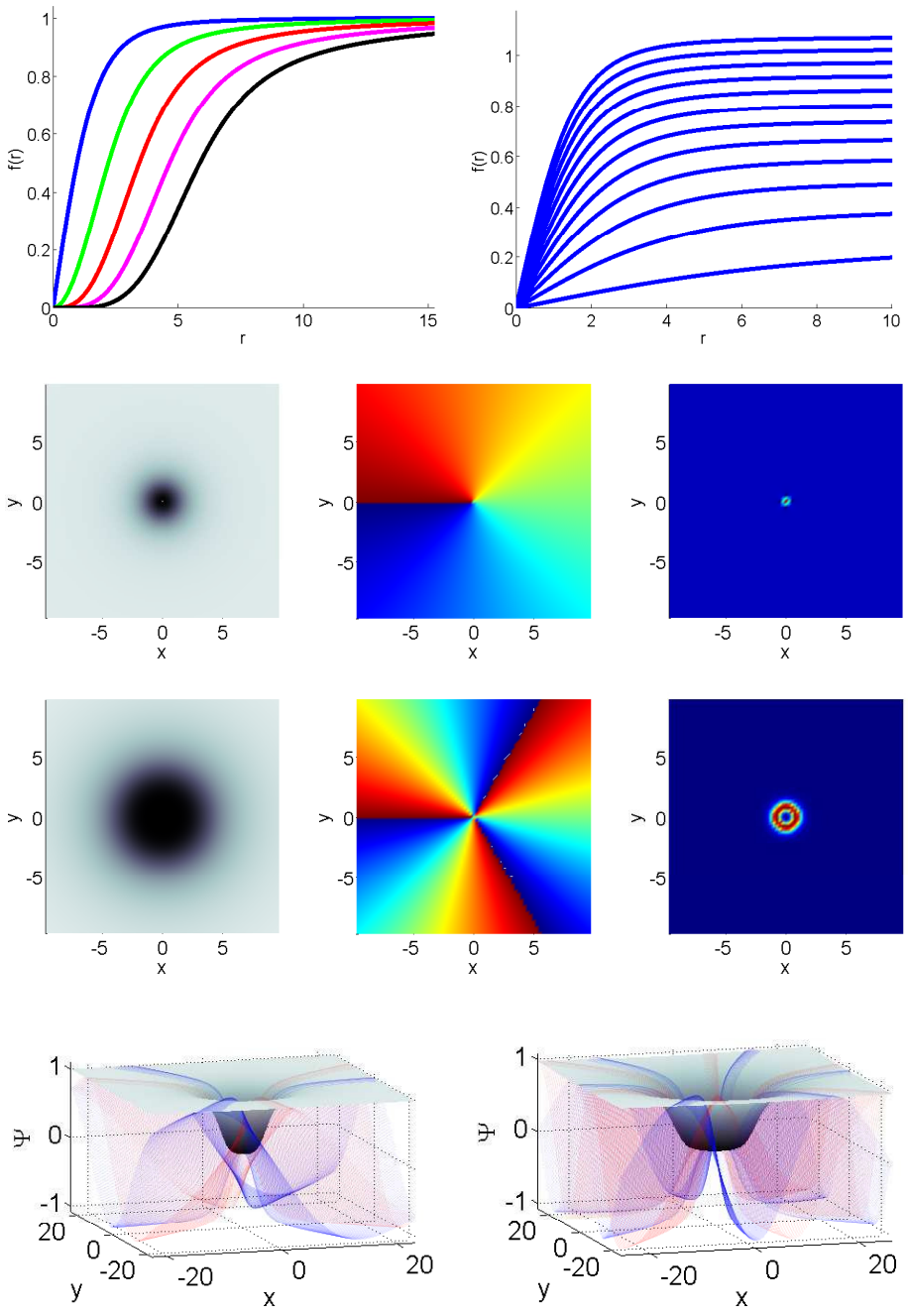


Figure 2.5. Depiction of dark vortices in the two-dimensional NLSE. Parameters for all plots are $a = 1$ and $s = -1$. Top left: Vortex profiles with $\Omega = -1$ and charges $m = 1 \to 5$ (left to right). Top right: Vortex profiles with charge $m = 1$ and frequencies $\Omega = -0.05 \to -1.2$ (bottom to top). The two middle rows show the density, phase, and vorticity (left to right) of a $m = 1$ (top) and $m = 3$ (bottom) vortex with $\Omega = -1$. The bottom row shows an $m = 3$ and $m = 6$ vortex with $\Omega = -1$. The blue and red mesh represents the real and imaginary parts of the wave-function respectively.

where x_i and y_i are the x and y positions of vortex i, r_{ik} is the distance between vortices i and k, b is a numerically derived constant ($b \approx 2$) and $i = 1...n$, where n is the total number of vortices.

Dark vortices of charge $|m| > 1$ are unstable [85] and break up into stable $|m| = 1$ charged vortices. This is understandable since the energy of a high-charge vortex is energetically unfavorable (as two well-separated vortices of charge $m = 1$ have less energy than one vortex of charge $m = 2$) [86]. An example of such a break-up for a charge $m = 3$ vortex is shown in Fig. 2.6. We see that the instability of a the vortex takes

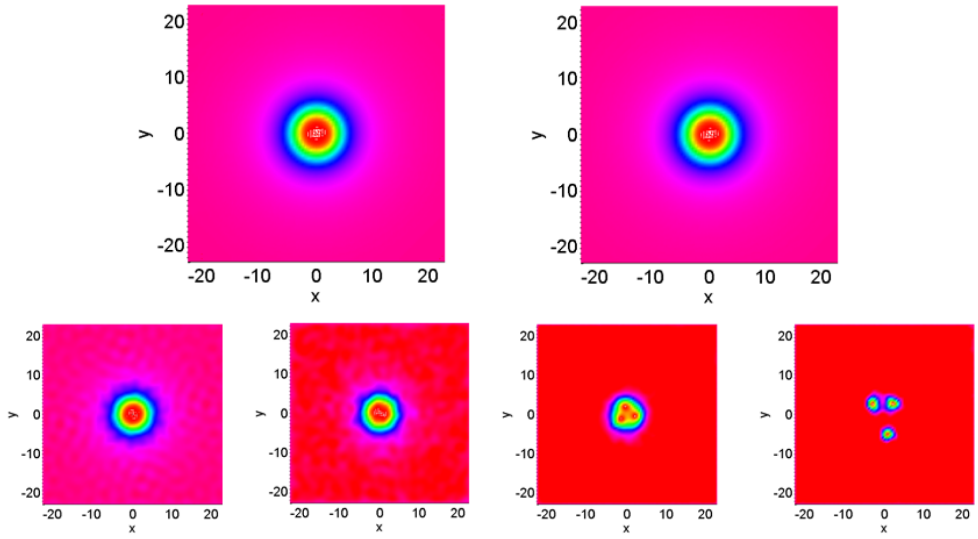


Figure 2.6. Depiction of break up of a charge $m = 3$ dark vortex in the two-dimensional NLSE. Top: The vortex at time $t = 0$ and $t = 50000$ with no perturbation. Bottom: The same vortex with a random perturbation of amplitude 0.01. The vortex is shown for times $t = 20000, 30000, 40000, 50000$. The NLSE parameters are $a = 1$, $s = -1$, and $\Omega = -1$, while the numerical parameters are $h = 0.5$ and $k = 0.058$ (see Chapter 3).

a long time to become apparent ($t > 50000$) unless a sizable perturbation is added to the solution. Therefore, although the dark vortices of higher charge are unstable, the growth rate of the instability appears to be quite low.

The interaction dynamics discussed in this section also should apply to higher-charge dark vortices. However, the combination and interference of the vortices

perturbs them (the closer the two vortices are to each other, the larger the perturbation) causing the instability of the high-charge vortices to becomes rapidly apparent. An example of this is shown in Fig. 2.7. It is clear that the super-position of two vortices

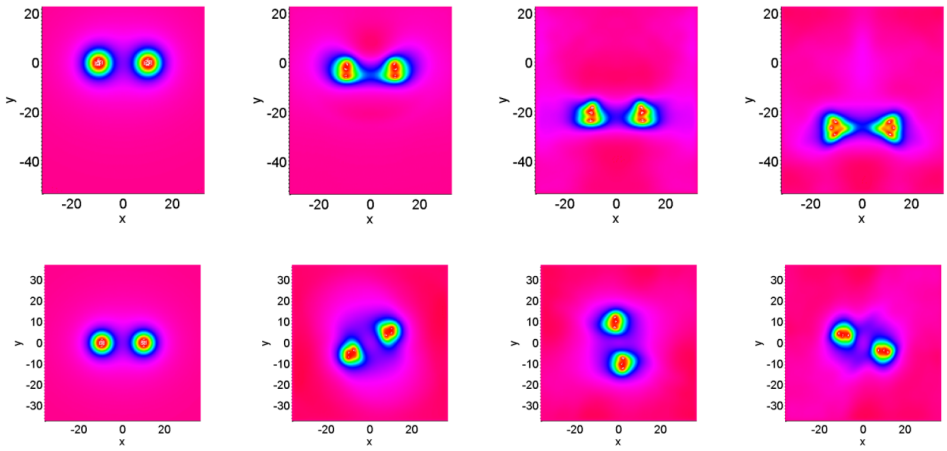


Figure 2.7. Depiction of the break-up of two charge $|m| = 3$ interacting dark vortices in the two-dimensional NLSE. Top: Two vortices of opposite charge separated by a distance of 20 at times $t = 0, 9, 69, 89$. Bottom: Two vortices of equal charge separated by the same distance at times $t = 0, 9, 19, 61, 96$. The NLSE and numerical parameters are the same as in Fig. 2.6.

causes them to break up into three charge $m = 1$ vortices much more quickly than in Fig. 2.6. From additional (not shown here) numerical simulations, we found that, as expected, the larger the separation between the vortices, the longer they last before visibly splitting.

2.4 Vortex Rings

A two-dimensional vortex extended into an infinite three-dimensional line is refereed to as a 'vortex line' [87, 88]. A 'vortex ring' resembles a vortex line which is bent into a ring making it axisymmetric. The resulting ring has an altered density profile and phase due to its self-interaction.

The most natural coordinate system to represent a vortex ring is axisymmetric cylindrical where the NLSE with $V(\mathbf{r}) = 0$ takes the form

$$i\,\frac{\partial \Psi}{\partial t} + a\left(\frac{1}{r}\frac{\partial \Psi}{\partial r} + \frac{\partial^2 \Psi}{\partial r^2} + \frac{\partial^2 \Psi}{\partial z^2}\right) + s\,|\Psi|^2\,\Psi = 0, \tag{2.30}$$

To describe the phase defect of the axisymmetric ring, a polar coordinate system is set up which is centered at the vortex ring radius (denoted as d) and its initial z position (z_0). The polar coordinates in the r–z plane are then given as

$$\phi(r,z) = \arctan\left(\frac{z-z_0}{r-d}\right), \tag{2.31}$$

$$\rho^2(r,z) = (r-d)^2 + (z-z_0)^2. \tag{2.32}$$

As we will discuss below, it is also useful to define a polar coordinate system which is centered off-axis at $r = -d$ as

$$\phi_2(r,z) = \arctan\left(\frac{z-z_0}{r+d}\right), \tag{2.33}$$

$$\rho_2^2(r,z) = (r+d)^2 + (z-z_0)^2. \tag{2.34}$$

A diagram of the coordinate systems to describe vortex rings is shown in Fig. 2.8.

2.4.1 Structure and Dynamics of Bright Vortex Rings

As mentioned in Chapter 1, a bright vortex ring's structure is described as a bright vortex wrapped around in the θ-direction forming a smoke-ring-like torus of density. A depiction of a charge $m = 2$ bright vortex ring with radius $d = 10$ is shown in Fig. 2.9.

In Sec. 2.3.2, it was mentioned that bright vortices in the NLSE are unstable and break-up into filaments which then collapse into singularities. This instability is expected to remain in three dimensions (which we confirm in Chapter 9). For this reason, bright vortex rings have not been studied at all as far as our literature review has found. We note

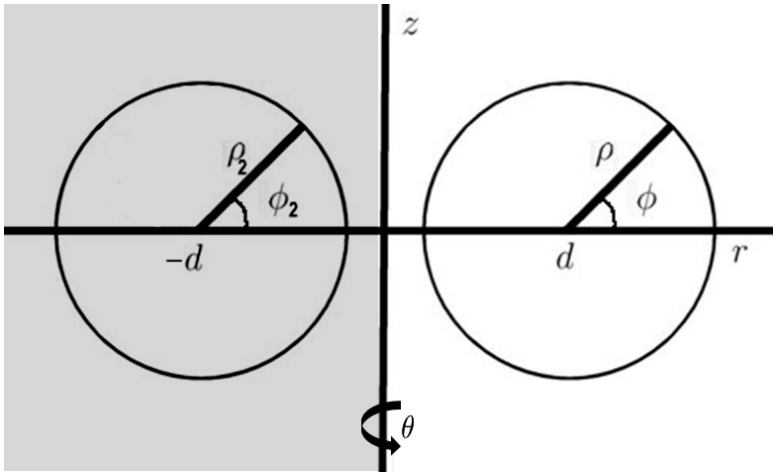


Figure 2.8. Coordinate system used to describe vortex rings. The ϕ_2 and ρ_2 variables are only used in the half plane with $r \geq 0$.

that three-dimensional bright 'spinning solitons' *have* been previously studied. These are bright vortices oriented in the x–y plane with a decaying density profile in the transverse z-direction. Such structures are also modulationally unstable, but can be stable in a cubic-quintic NLSE setting [89].

Since the density of the bright rings trail off to zero rapidly away from the tube edge, the phase interaction of one side of the ring to the opposite size is minimal and therefore the ring is expected (unlike the dark vortex rings discussed below) to be nearly steady-state with no transverse velocity. Therefore, the axisymmetric solution $\Psi(r, z, 0)$ at $t = 0$ of the vortex ring can be approximately found as a solution to the steady-state PDE

$$-\Omega\,\Psi + a\left(\frac{1}{r}\frac{\partial\Psi}{\partial r} + \frac{\partial^2\Psi}{\partial r^2} + \frac{\partial^2\Psi}{\partial z^2}\right) + s\,|\Psi|^2\,\Psi = 0,$$

where $\Psi(r, z, 0) = g(r, z)\exp[i\,m\,\phi]$, where $g(r, z)$ is the density profile of the vortex ring's axisymmetric cut (taken to be the density profile of a bright two-dimensional vortex). Numerical simulations of the collapse of bright vortex rings are shown in Chapter 9.

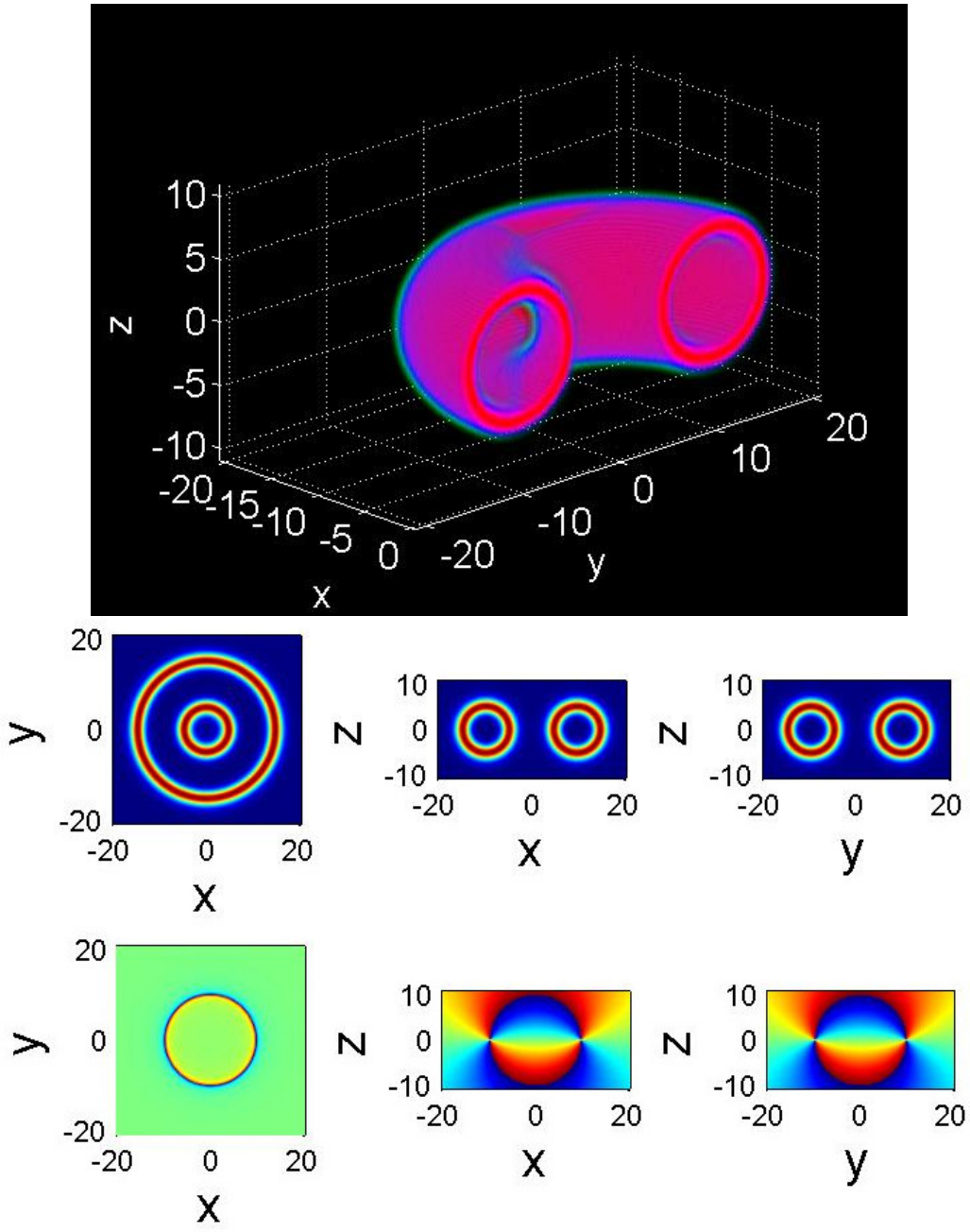


Figure 2.9. Depiction of a bright vortex ring of charge $m = 2$, radius $d = 10$ and frequency $\Omega = 1/3$ in the three-dimensional NLSE. Top: Volumetric rendering of half the vortex ring showing the internal structure. Middle: Density of two-dimensional cuts. Bottom: Phase of two-dimensional cuts.

2.4.2 Structure and Dynamics of Dark Vortex Rings

As mentioned in Chapter 1, dark vortex rings are a ring-shaped line of zero-density with increasing density in the ρ-direction converging to a constant infinite background density. As was the case with the dark vortices, the background density is given as $\Psi_\infty = \Omega/s$.

A depiction of a charge $m = 1$ dark vortex ring with radius $d = 10$ is shown in Fig. 2.10. Due to the structure of the dark vortex rings, it can be difficult to visualize them in three dimensions. Therefore, we visualize them using a volumetric rendering with a transparency map which shows constant density as fully transparent and scales so that zero-density is fully opaque.

As mentioned in Chapter 1, since the phase of dark structures in the NLSE 'see' each other instantaneously over long distances, a dark vortex ring will have an intrinsic transverse velocity which we denote as c. This can be understood by noting that an axisymmetric cut of a dark vortex ring is qualitatively similar to two dark vortices of opposite charge, which as shown in Sec. 2.3.3, travel in parallel with a constant velocity.

The value of c for dark vortex rings with charge $m = 1$ has been studied analytically using various approximation techniques [36, 37]. In Ref. [15], a thorough asymptotic analysis was performed yielding an expression of the vortex ring's velocity which is accurate to $O((r_c/d)^2)$ where r_c is taken to be the vortex ring's core radius defined as

$$r_c = |m|\sqrt{-\frac{a}{\Omega}}, \tag{2.35}$$

which is equivalent to that of a dark vortex. The velocity result relies on a constant value found through numerical integration called the 'vortex core parameter' denoted as L_0. The vortex core parameter was first derived and computed in Ref. [90] (although not as an explicit separate term) for charges $m = 1, 2, 3$ where it represents the non-dimensionalized convergent part of the energy per unit length of a three-dimensional

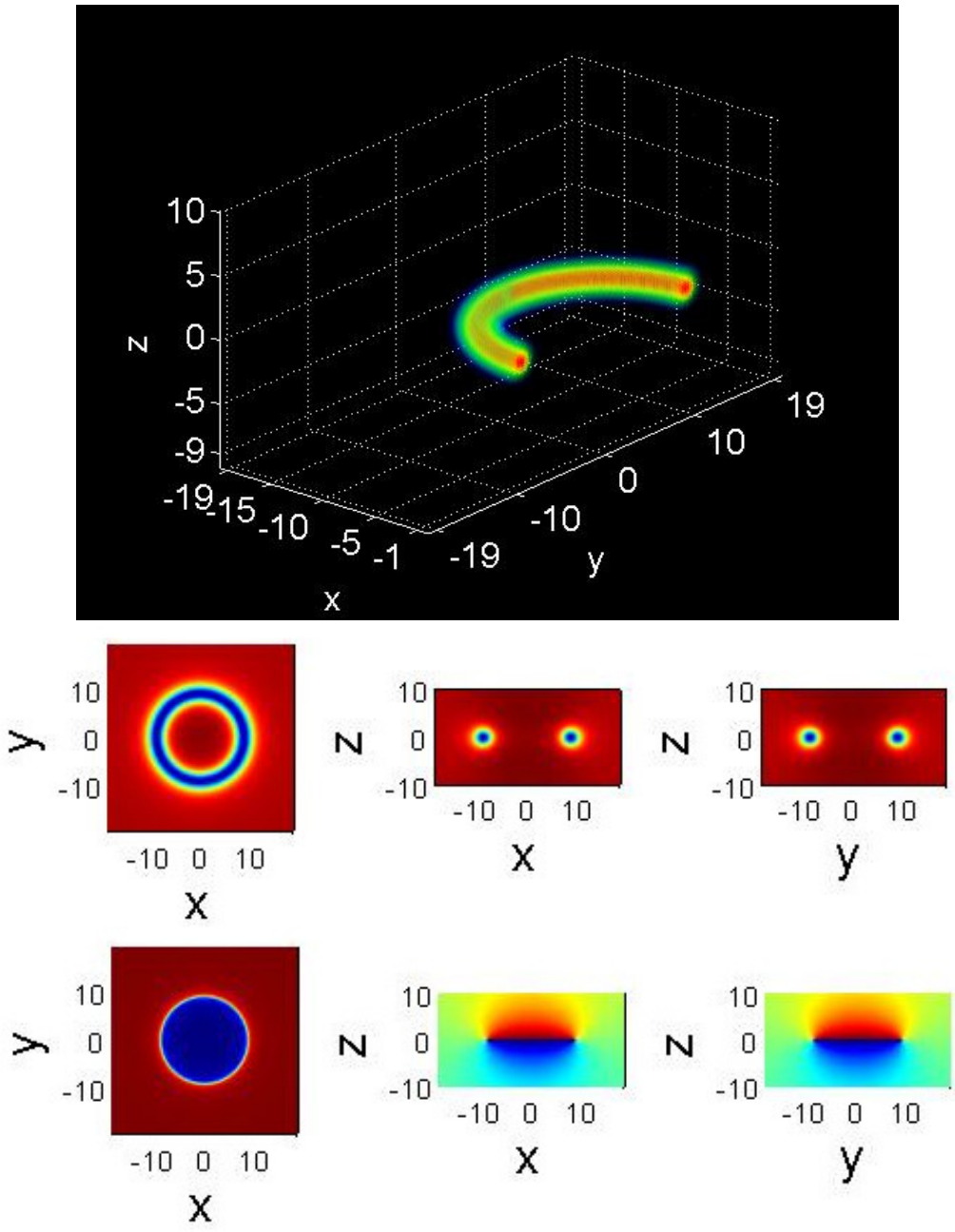


Figure 2.10. Depiction of a dark vortex ring of charge $m = 1$, radius $d = 10$ and frequency $\Omega = -1$ in the three-dimensional NLSE. Top: Volumetric rendering of half of the vortex ring where zero-density is shown as fully opaque, while the background density is fully transparent. Middle: Density of two-dimensional cuts. Bottom: Phase of two-dimensional cuts.

infinite-extent dark vortex line. The values of $L_0(m)$ for $m = 1, 2, 3$ (in terms of the form chosen in Appendix A) were given as $\ln(1.46) = 0.3784$, $\ln(0.59 * 2) = 0.1655$, and $\ln(0.38 * 3) = 0.1310$ respectively. In Ref. [15], L_0 for charge $m = 1$ was recomputed as $L_0(1) = 0.3850$, while in Ref. [91] (and others) it is reported as $L_0(1) = 0.3809$.

Since we would like to have the velocity of vortex rings with charge $|m| > 1$ (even though they are energetically unfavorable, see Ref. [86]), and there has been disagreement in the values of L_0, we have re-derived the form of the constant L_0 for any charge and computed their values in Appendix A. The computed L_0 values for $m = 1 \rightarrow 10$ are given in Table. 2.1

Table 2.1. Values of the vortex core parameter L_0 for charges $m = 1 \rightarrow 10$. The values are computed through numerical integration as described in Appendix A

m	$L_0(m)$	m	$L_0(m)$
1	0.380868	6	0.022954
2	0.133837	7	0.017785
3	0.070755	8	0.014247
4	0.044567	9	0.011713
5	0.030981	10	0.009833

.

As mentioned in Appendix A, the L_0 data can be fitted to yield the relationship

$$L_0(m) \approx \frac{L_0(1)}{m^{1.6}}, \tag{2.36}$$

which is very close to the computed values of L_0 as shown in Fig. 2.11.

The energy per unit length (see Appendix A) of a quantized vortex line is given as

$$\mathcal{E} = \frac{2\,\pi\,a\,m^2\,\Omega}{s}\left[\ln\left(\frac{L}{r_c}\right) + L_0(m)\right], \tag{2.37}$$

where L is a cut-off parameter required to make the energy integral convergent. Through an asymptotic analysis, the authors of Ref. [15] determined that the energy of a vortex ring can be given simply by the intuitive $E = 2\,\pi\,d\,\mathcal{E}$ if the cut-off distance L is chosen as

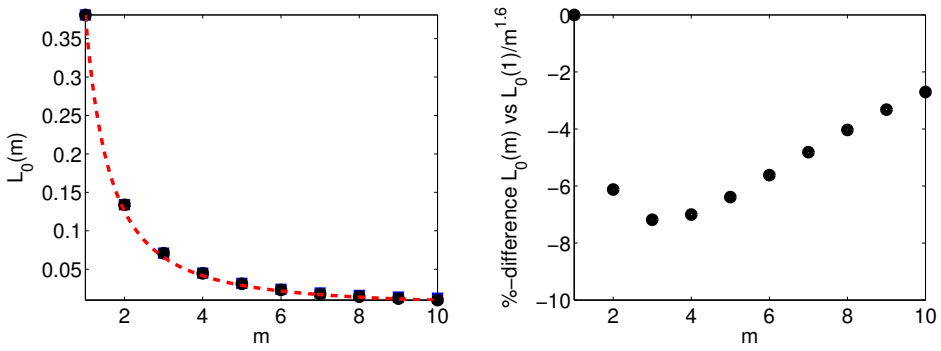


Figure 2.11. Left: Vortex core parameters for charges $m = 1$ to $m = 10$ using the computational methodology described in Appendix A. The (blue) squares were found using Eq. (A.9) while the black dots used Eq. (A.8). The (red) dashed line is the fitted curve of Eq. (2.36). Right: Percent-difference between the $L_0(m)$ results obtained from Eq. (A.8) and the fitted curve of Eq. (2.36).

$L = 8d/e^2$ (a term deriving from equating the first two terms in the asymptotic expansion of the energy of a classic vortex ring in fluid mechanics [92] to the first term in Eq. (2.37)), yielding an energy of

$$\mathcal{E} = \frac{4\,\pi^2\,a\,d\,m^2\,\Omega}{s}\left[\ln\left(\frac{8\,d}{r_c}\right) + L_0(m) - 2\right]. \tag{2.38}$$

It was further determined that the Hamiltonian relationship

$$c = \frac{\partial E}{\partial p},$$

holds where p is the impulse given by

$$p = \frac{\Omega}{s}\,2\,m\,\pi^2\,d^2.$$

Evaluating $c = (\partial E/\partial d)(\partial d/\partial p)$ then yields the expression of the vortex ring velocity given as [15]

$$c \approx -\frac{a\,m}{d}\left[\ln\left(8\,d\sqrt{-\frac{\Omega}{a\,m^2}}\right) + L_0(m) - 1\right]. \tag{2.39}$$

In Sec. 9.1.2, Eq. (2.39) is compared to direct numerical simulations of dark vortex rings of various radii and shown to be extremely accurate.

As we will show in Chapter 9, since the dark vortex rings move at a constant velocity, in order to numerically formulate the structure of the vortex rings, it is useful to view them in a co-moving reference frame. In Appendix B, a derivation of a general form of a co-moving solution to the NLSE is shown. In the context of vortex rings aligned with the x–y plane with transverse velocity c in the z-direction, the co-moving solution derived in Appendix B in cylindrical coordinates is given by

$$\Psi(r,z,\theta,t) = U(r,Z,\theta)\exp\left[i\left(\frac{c}{2\,a}\,z + \left[\Omega - \frac{c^2}{4\,a}\right]\,t\right)\right], \tag{2.40}$$

where $Z = z - c\,t$ and $U(r,Z,\theta)$ solves the time-independent NLSE

$$-\Omega\,U + a\,\nabla^2 U + s\,|U|^2\,U = 0. \tag{2.41}$$

As in the case of bright vortex rings, a standard form of the initial condition of a dark vortex ring solution is

$$\Psi(r,\theta,z,0) = g(r,z)\exp[i\,m\,\phi(r,z)]. \tag{2.42}$$

but in this case, due to the long-range self-interaction of the ring, the phase structure of the vortex ring is not symmetric over ϕ, and therefore g cannot be assumed to be real-valued. In order to find the structure of $g(r,z)$, the initial condition of Eq. (2.42) is inserted into Eq. (2.40), and since when $t = 0$, $Z = z$, the initial condition becomes

$$U_0 = \Psi_0\exp\left(-i\frac{c}{2a}z\right). \tag{2.43}$$

Inserting Eq. (2.43) into Eq. (2.41) in cylindrical coordinates yields a time-independent PDE governing g given by

$$-\left[\Omega+\frac{c^2}{4\,a}+\frac{a\,m^2}{\rho^2}-(r-d)\frac{m\,c}{\rho^2}\right]\,g+a\left[\frac{1}{r}\,g_r+g_{rr}+g_{zz}\right]+s\,|g|^2\,g+ \\ i\,\frac{a\,m}{\rho^2}\left[2\,(r-d)\;g_z-(z-z_0)\left(2g_r+\frac{1}{r}\,g\right)\right]-i\,c\,g_z=0. \tag{2.44}$$

A similar PDE to Eq. (2.44) was formulated for the NLSE with an external trapping potential in Ref. [16], where steady-state vortex ring solutions were found using imaginary time integration.

Since the phase structure in the axisymmetric cut of a vortex ring will have the interference of the vortex core at $r=d$ and that at $r=-d$, in order to more accurately describe the phase of the vortex ring in the half plane (making $g(r,z)$ contain less phase correction), we reformulate the initial condition to be

$$\Psi_0(r,\theta,z)=g(r,z)\exp[i\,m\,(\phi(r,z)-\phi_2(r,z))], \tag{2.45}$$

yielding a more complicated PDE governing the structure of of g given by

$$-\left[\Omega+\frac{c^2}{4\,a}-\frac{4\,m}{\rho_1^2\rho_2^2}\left(a\,m((z-z_0)^2+r^2)+c\,r^3+c\,r\,(z-z_0)^2\right)\right. \\ \left.+\;\frac{m}{\rho_1^2}(2\,a\,m+r\,c)+\frac{m}{\rho_2^2}(2\,a\,m+5\,c\,r-2\,c(r-d))\right] \\ +\,a(g_{rr}+g_{zz})+\left(\frac{4ar}{\rho_2^2}-\frac{4a(r-d)}{\rho_2^2}+\frac{a\rho_1^2}{r\rho_2^2}\right)\,g_r+s\,|g|^2\,g \\ +\,i\left[2\,a\,m(z-z_0)\,g_r\left(\frac{1}{\rho_2^2}-\frac{1}{\rho_1^2}\right)-\frac{4amd(z-z_0)}{\rho_1^2\rho_2^2}\,g\right. \\ \left.+\left(\frac{8am+8amr(z-z_0)^2}{\rho_1^2\rho_2^2}-\frac{6amr+4amd}{\rho_2^2}-\frac{2amr}{\rho_1^2}-c\right)\,g_z\right]=0. \tag{2.46}$$

Although Eq. (2.46) is cumbersome to handle and therefore not utilized in the present study, as we will show in Chapter 9, the initial condition structure of Eq. (2.45) yields a

much closer vortex solution when combined with the two-dimensional dark vortex density profile than using Eq. (2.42).

We leave the details of the generation of numerically-exact vortex ring solutions to Chapter 9. We now leave the discussion of vortex rings to describe the numerical methodology and code implementations we will use to study the dynamics of the rings.

CHAPTER 3

NUMERICAL ALGORITHMS: EXPLICIT TWO-STEP HIGH-ORDER COMPACT SCHEMES

As described in Sec. 1.1.3, when implementing high-order finite-difference schemes in a parallel environment, it is beneficial to have the schemes to be compact. In addition, although most time-dependent high-order compact (HOC) schemes are implicit, for large, higher-dimensional problems it is desirable to have an explicit scheme to avoid having to solving large linear systems with iterative techniques to handle nonlinearities. Since we plan to implement our numerical schemes for the NLSE into three-dimensional GPU-accelerated codes, we wish to develop HOC formulations for the Laplacian operator $\left(\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right)$ that are *fully explicit*. To do this, we formulate a two-step procedure in computing the Laplacian's spatial derivatives, where each individual step is a compact computation thus forming a two-step high-order compact (2SHOC) scheme. The first step computes the standard second-order finite-difference approximation to the derivatives, while the second step uses those results to compactly approximate the Laplacian to fourth-order accuracy. Since the 2SHOC scheme computes the Laplacian independent of the governing PDE, it can be used in any PDE which contains the Laplacian operator, and is therefore quite general. An explicit time-dependent 2SHOC scheme for the NLSE can then be formed by combining the 2SHOC scheme with a standard explicit ordinary differential equation (ODE) solver.

3.1 FOURTH-ORDER RUNGE-KUTTA

To formulate a fully explicit scheme for simulating the NLSE, a method of lines approach is utilized, where the 2SHOC is used for the spatial Laplacian and the resulting

semi-discrete system of ODEs is integrated using an explicit time-stepping scheme. As shown in Ref. [93] and Chapter 5, the first-order ($O(\mathbf{k})$ where $\mathbf{k}$ is the time-step) forward difference and the second-order ($O(\mathbf{k^2})$) Runge-Kutta scheme (Heun's method) for simulating the NLSE are unconditional *unstable* (unless, as shown in Ref. [94], the real and imaginary parts of the NLSE are computed in staggered time steps). Therefore, for our implementation, a higher-order Runge-Kutta method is required and is chosen to be the standard fourth-order Runge-Kutta (RK4) scheme [95]. To facilitate our computational cost comparisons in Sec. 3.4, we write the RK4 scheme applied to the NLSE algorithmically as

$$
\begin{aligned}
&1)\ K_{\text{tot}} = F(\Psi^n) && 6)\ K_{\text{tmp}} = F(\Psi_{\text{tmp}}) \\
&2)\ \Psi_{\text{tmp}} = \Psi^n + \frac{\mathbf{k}}{2} K_{\text{tot}} && 7)\ K_{\text{tot}} = K_{\text{tot}} + 2\, K_{\text{tmp}} \\
&3)\ K_{\text{tmp}} = F(\Psi_{\text{tmp}}) && 8)\ \Psi_{\text{tmp}} = \Psi^n + \mathbf{k}\, K_{\text{tmp}} \\
&4)\ K_{\text{tot}} = K_{\text{tot}} + 2\, K_{\text{tmp}} && 9)\ K_{\text{tmp}} = F(\Psi_{\text{tmp}}) \\
&5)\ \Psi_{\text{tmp}} = \Psi^n + \frac{\mathbf{k}}{2} K_{\text{tmp}} && 10)\ \Psi^{n+1} = \Psi^n + \frac{\mathbf{k}}{6} (K_{\text{tot}} + K_{\text{tmp}}),
\end{aligned}
\tag{3.1}
$$

where

$$
F(\Psi) = \frac{\partial \Psi}{\partial t} = i \left[a \nabla^2 \Psi + \left(s \left| \Psi \right|^2 - V(\mathbf{r}) \right) \Psi \right],
$$

and $\nabla^2 \Psi$ inside $F(\Psi)$ is computed with either a standard central differencing (CD) or the 2SHOC scheme (defined in the next section). We denote the combined time-space schemes as RK4+CD and RK4+2SHOC respectively.

The RK4 time-stepping requires three storage arrays (K_{tmp}, K_{tot}, and Ψ_{tmp}) in addition to the storage for the solution Ψ and external potential $V(\mathbf{r})$ (where $V(\mathbf{r})$ is chosen to be stored as an array of values rather than computed at each evaluation). Although lower-storage fourth-order Runge-Kutta schemes with comparable accuracy have been developed (a $3N$ in Ref. [96], and a five-stage $2N$ in Ref. [97] which requires

an additional function evaluation), for simplicity we use the classic RK4 of Eq. (3.1). Using the 2SHOC scheme inside $F(\Psi)$ requires additional storage, which is discussed in Sec. 3.4.

The RK4+CD and RK4+2SHOC schemes are conditionally stable, in that the size of the time-step is limited by a bound based on the spatial step-size h (see Chapter 5, where approximations to these stability requirements are derived). This conditional stability is one of the only drawbacks of using explicit schemes. However, even though implicit schemes are usually unconditionally stable, error requirements and algorithm complexity can make the explicit schemes more efficient, even when considering the time-step limitations. A thorough study comparing the efficiency of equivalently-accurate explicit and implicit schemes for simulating the NLSE is beyond the scope of this work. However, we do note that since many implicit schemes (such as the widely-used Crank-Nicholson [98]) are only second-order accurate in time, if the same order of accuracy is sought for both the time and spatial errors, then using a $O(\mathbf{k}^2) + O(h^4)$ implicit method will require a time-step which is of the same order ($\mathbf{k} \sim O(h^2)$) as the time-step required for the stability of the RK4+2SHOC.

3.2 Formulation of 2SHOC Schemes for the Laplacian Operator

It is well known that one can derive a fourth-order accurate finite-difference scheme for the second spatial derivative of a function by applying a 'double-differencing' approach. This works by noting that the first truncation term in the standard central-difference scheme contains the fourth spatial derivative. A second-order approximation to the fourth derivative can be obtained by applying a central-difference operator to the result of applying a central-difference operator to the function. When multiplied by the h^2 in the truncation term, the error in the truncation term becomes $O(h^4)$, and therefore, the resulting scheme becomes a fourth-order accurate approximation to the second derivative. When these two steps are algebraically combined

and simplified, one yields the standard non-compact fourth-order stencil. Due to the increase in the number of computations and storage, the 'double-differencing' procedure is not actually implemented, rather, the resulting non-compact stencil is used directly.

However, when the advantages of using a compact scheme are considered, the numerical implementation of the 'double-differencing' procedure becomes an overlooked, viable alternative to other more complicated HOC schemes. In time-dependent systems, the sequential nature of the 'double-differencing' procedure makes it best suited for use with explicit time-stepping schemes. This may be a reason why this simple compact solution has been overlooked, as many numerical schemes for the NLSE in common use are typically implicit.

3.2.1 One-Dimensional Formulation

Although the methodology for the 2SHOC scheme is to use the 'double-differencing' approach discussed in the previous section, it is nevertheless worthwhile to show that the scheme can also be formed as a result of attempting to make other time-dependent HOC scheme formulations explicit. In Ref. [60], the authors formulate an implicit time-dependent HOC scheme for the transient convection-diffusion equation based on their method from Ref. [54]. We can obtain a compact high-order scheme for the time-dependent NLSE following the same methodology, and with a modification, can transform it into an explicit scheme which is equivalent to the 2SHOC.

In one dimension, we discretize the wavefunction Ψ of the NLSE as $\Psi(x_i, t_n) \equiv \Psi_i^n$ with a grid spacing of size h ($x_i = x_{\text{min}} + hi$) and a time-step of size $\mathbf{k}$ ($t_n = \mathbf{k}\, n$). The Laplacian operator applied to Ψ in one dimension at grid-point i can then be represented as

$$\nabla^2 \Psi_i = \left.\frac{\partial^2 \Psi}{\partial x^2}\right|_i = \delta_x^2 \Psi_i - \frac{h^2}{12}\left(\left.\frac{\partial^4 \Psi}{\partial x^4}\right|_i\right) + O(h^4), \tag{3.2}$$

where the central-difference operator δ_x^2 is defined as

$$\delta_x^2 \, \Psi_i = \frac{\Psi_{i+1} - 2\,\Psi_i + \Psi_{i-1}}{h^2}. \tag{3.3}$$

As per the technique of Ref. [60], to formulate a high-order compact scheme, the NLSE of Eq. (1.2) is differentiated in space twice in order to form an expression for the fourth derivative in Eq. (3.2) which yields

$$\left.\frac{\partial^4 \Psi}{\partial x^4}\right|_i = -\frac{1}{a}\,\delta_x^2 \left[i \left.\frac{\partial \Psi}{\partial t}\right|_i + \left(s|\Psi_i|^2 - V(x_i) \right) \Psi_i \right] + O(h^2). \tag{3.4}$$

Due to the h^2 in the truncation term of Eq. (3.2), when Eq. (3.4) is inserted into Eq. (3.2), the resulting approximation of the Laplacian becomes $O(h^4)$. Inserting Eqs. (3.2) and (3.4) into Eq. (1.2) yields the fourth-order in space semi-discrete HOC scheme

$$\begin{aligned} \left.\frac{\partial \Psi}{\partial t}\right|_i =& i[a \left(\delta_x^2 \, \Psi_i + \frac{h^2}{a\,12} \delta_x^2 \left[i \left.\frac{\partial \Psi}{\partial t}\right|_i + \left(s\,|\Psi_i|^2 - V(x_i) \right) \Psi_i \right]_i \right) \\ & + \left(s\,|\Psi_i|^2 - V(x_i) \right) \Psi_i]. \end{aligned} \tag{3.5}$$

Due to the central-difference operator operating on the temporal derivative $\partial\Psi/\partial t$, the resulting scheme of Eq. (3.5) will be implicit even if the time-stepping is chosen to be otherwise explicit (for example, forward differencing).

In order to retain a fully explicit scheme, a two-step approach is used. First, an approximation of $\partial\Psi_i/\partial t$ is made and then is inserted into Eq. (3.5). The approximation of $\partial\Psi_i/\partial t$ must be $O(h^2)$ for the scheme to retain its fourth-order accuracy. The most straight-forward way to make such an approximation is to apply a standard second-order central differencing to the NLSE yielding

$$T_i = \left.\frac{\partial \Psi}{\partial t}\right|_i + O(h^2) = i \left[a\delta_x^2 \, \Psi_i + \left(s\,|\Psi_i|^2 - V(x_i) \right) \Psi_i \right]. \tag{3.6}$$

Once computed, T_i is inserted into Eq. (3.5), and the central difference operator can then be applied to T_i as

$$\delta_x^2 T_i = \frac{T_{i+1} - 2\,T_i + T_{i-1}}{h^2}.$$

It is important that the boundary conditions for Eq. (3.6) be computed to at least $O(h^2)$ accuracy since the boundary points will be used for interior calculations of the spatial derivative of Eq. (3.5). (see Chapter 4 for details on the implementation of boundary conditions for the 2SHOC schemes). Inserting Eq. (3.6) into Eq. (3.5) yields the semi-discrete equation

$$\begin{aligned}\left.\frac{\partial \Psi}{\partial t}\right|_i =& i[a\left(\delta_x^2 \Psi_i + \frac{h^2}{a\,12}\delta_x^2\left[i\,T_i + \left(s\,|\Psi_i|^2 - V(x_i)\right)\Psi_i\right]_i\right) \\ &+ \left(s\,|\Psi_i|^2 - V(x_i)\right)\Psi_i] + O(h^4).\end{aligned} \tag{3.7}$$

Any desired explicit ODE solver (that is stable for the problem with its parameters) can then be used to integrate Eq. (3.7).

Algebraically combining the two stages of Eq. (3.6) and Eq. (3.7) together yields

$$\left.\frac{\partial \Psi}{\partial t}\right|_i = i\left[a\left(\delta_x^2 \Psi_i - \frac{h^2}{12}\delta_x^2(\delta_x^2 \Psi_i)\right) + \left(s\,|\Psi_i|^2 - V(x_i)\right)\Psi_i\right] + O(h^4), \tag{3.8}$$

and therefore the two-stage HOC scheme described in Eq. (3.6) and Eq. (3.7) is computationally equivalent to simply taking the central-difference of the central-difference to approximate the fourth derivative truncation term in the Laplacian. Therefore, we have recovered the 2SHOC scheme approach for approximating the Laplacian which does not depend on the other terms of the governing equation. Therefore the 2SHOC is a stand-alone scheme for the Laplacian operator which can be used in multiple governing equations.

After collecting terms and simplifying the 2SHOC 'double-differencing', the resulting two-step scheme for the one-dimensional Laplacian is given by

$$1) \qquad D_i = \frac{1}{h^2}\left(\Psi_{i+1} - 2\Psi_i + \Psi_{i-1}\right), \tag{3.9}$$

$$2) \qquad \nabla^2\Psi_i \approx \frac{7}{6}D_i - \frac{1}{12}\left(D_{i+1} + D_{i-1}\right). \tag{3.10}$$

When the two steps of Eqs. (3.9) and (3.10) are combined algebraically and simplified, the standard five-point non-compact fourth-order finite-difference approximation is recovered (see Appendix B for details):

$$\nabla^2\Psi_i = \left.\frac{\partial^2\Psi}{\partial x^2}\right|_i = \frac{-\Psi_{i+2} + 16\Psi_{i+1} - 30\Psi_i + 16\Psi_{i-1} - \Psi_{i-2}}{12h^2} + O(h^4).$$

A potential drawback in using the 2SHOC scheme is that it requires extra storage space (the D array) and more computations than using the standard fourth-order five-point stencil. However, as will be discussed in Sec. 3.4, the compact scheme's advantages can outweigh this deficiency.

3.2.2 Two-Dimensional Formulation

In two dimensions, the Laplacian operator applied to Ψ at grid location (i, j) can be represented as

$$\nabla^2\Psi_{i,j} = \frac{1}{h^2}\begin{array}{|c|c|c|}\hline & 1 & \\ \hline 1 & \text{-4} & 1 \\ \hline & 1 & \\ \hline\end{array}\Psi_{i,j} - \frac{h^2}{12}\left(\left.\frac{\partial^4\Psi}{\partial x^4}\right|_{i,j} + \left.\frac{\partial^4\Psi}{\partial y^4}\right|_{i,j}\right) + O(h^4). \tag{3.11}$$

Unlike the one-dimensional case, there are additional compact grid points which are not being used in Eq. (3.11) (the four corner points). It is known that these points can be

added to make a more accurate nine-point Laplacian operator given as [99]

$$\nabla^2\Psi_{i,j} = \frac{1}{6\,h^2}\begin{array}{|c|c|c|}\hline 1 & 4 & 1 \\ \hline 4 & \text{-}20 & 4 \\ \hline 1 & 4 & 1 \\ \hline \end{array}\,\Psi_{i,j} + O(h^2). \tag{3.12}$$

However, the nine-point Laplacian of Eq. (3.12), while more accurate, is still second-order. In fact, it can easily be shown (see Appendix D for proof) that there *cannot* exist a fourth-order nine-point Laplacian operator. (This fact should not be confused with the well-known fourth-order nine-point scheme for the Laplace and Poisson equations [66], which rely on multiplying the entire equation by h^2, or on taking finite differences of the right-hand side function respectively to eliminate the additional error terms). Therefore, even with the added compact grid points, the 2SHOC scheme is still required for fourth-order accuracy. However, since the nine-point Laplacian of Eq. (3.12) is still only second-order, in order to minimize the amount of computation needed for the 2SHOC scheme, we will only utilize the standard five-point Laplacian of Eq. (3.11) for the required second-order derivatives. Once again, 'double-differencing' is used to obtain a second-order accurate approximation for the truncation term's fourth derivatives. However, in this case, the result yields two variations of the 2SHOC scheme.

The first is a direct parallel to the one-dimensional 2SHOC scheme in which the second derivative in the x and y directions are approximated with second-order central-differencing. These are then used to form second-order approximations to the fourth derivatives in the truncation terms in Eq. (3.11). After simplification, this results in

the 2SHOC scheme

$$1) \qquad D^x_{i,j} = \delta^2_x \Psi_{i,j} = \frac{\Psi_{i+1,j} - 2\Psi_{i,j} + \Psi_{i-1,j}}{h^2}, \tag{3.13}$$

$$D^y_{i,j} = \delta^2_y \Psi_{i,j} = \frac{\Psi_{i,j+1} - 2\Psi_{i,j} + \Psi_{i,j-1}}{h^2},$$

$$2) \qquad \nabla^2 \Psi_{i,j} \approx \frac{7}{6}\left(D^x_{i,j} + D^y_{i,j}\right) - \frac{1}{12}\left(D^x_{i+1,j} + D^x_{i-1,j} + D^y_{i,j+1} + D^y_{i,j-1}\right). \tag{3.14}$$

The 2SHOC scheme as described in Eqs. (3.13) and (3.14) require two storage arrays (D_x and D_y) in addition to that for Ψ. In large-scale computations, memory use can be a major bottleneck and so it is useful to limit the amount of extra storage required as much as possible. We find that the two-dimensional 2SHOC scheme can be re-formulated to only require *one* extra storage matrix, at the cost of requiring more computations (a trade-off that is evaluated in Sec. 3.4).

To formulate the lower-storage version of the two-dimensional 2SHOC, the standard five-point second-order finite-difference approximation to the two-dimensional Laplacian is used for the first stage (stored in array D), and then the result is combined with a second-order cross-derivative stencil to yield the second-order approximations to the fourth derivative truncation terms in Eq. (3.11). First, we note that

$$\nabla^2\left(\nabla^2\Psi\right) = \frac{\partial^2}{\partial x^2}\nabla^2\Psi + \frac{\partial^2}{\partial y^2}\nabla^2\Psi = \frac{\partial^4\Psi}{\partial x^4} + \frac{\partial^4\Psi}{\partial y^4} + 2\frac{\partial^4\Psi}{\partial x^2\,\partial y^2},$$

in which case the fourth derivative truncation terms in Eq. (3.11) can be written as

$$\frac{\partial^4\Psi}{\partial x^4} + \frac{\partial^4\Psi}{\partial y^4} = \frac{\partial^2}{\partial x^2}\nabla^2\Psi + \frac{\partial^2}{\partial y^2}\nabla^2\Psi - 2\frac{\partial^4\Psi}{\partial x^2\,\partial y^2}. \tag{3.15}$$

The cross-derivative in Eq. (3.15) is known to have the nine-point second-order compact stencil [100]

$$\left.\frac{\partial^4\Psi}{\partial x^2\,\partial y^2}\right|_{i,j} = \frac{1}{h^4}\begin{array}{|c|c|c|}\hline 1 & -2 & 1 \\ \hline -2 & 4 & -2 \\ \hline 1 & -2 & 1 \\ \hline \end{array}\,\Psi_{i,j} + O(h^2). \tag{3.16}$$

Eq. (3.15) can now be computed to second-order accuracy as

$$\frac{\partial^4\Psi}{\partial x^4} + \frac{\partial^4\Psi}{\partial y^4} = \frac{1}{h^2}\begin{array}{|c|c|c|}\hline & 1 & \\ \hline 1 & -4 & 1 \\ \hline & 1 & \\ \hline \end{array}\,D_{i,j} - \frac{2}{h^4}\begin{array}{|c|c|c|}\hline 1 & -2 & 1 \\ \hline -2 & 4 & -2 \\ \hline 1 & -2 & 1 \\ \hline \end{array}\,\Psi_{i,j} + O(h^2), \tag{3.17}$$

where $D_{i,j} = \delta_x^2\Psi_{i,j} + \delta_y^2\Psi_{i,j}$ is the second-order approximation to the Laplacian. Part of the Ψ stencil of Eq. (3.17) can be written in terms of D, in which case the single-storage 2SHOC scheme (after simplification) is given as

$$1)\quad D_{i,j} = \delta_x^2\Psi_{i,j} + \delta_y^2\Psi_{i,j} = \frac{1}{h^2}\begin{array}{|c|c|c|}\hline & 1 & \\ \hline 1 & -4 & 1 \\ \hline & 1 & \\ \hline \end{array}\,\Psi_{i,j} \tag{3.18}$$

$$2)\quad \nabla^2\Psi_{i,j} \approx -\frac{1}{12}\begin{array}{|c|c|c|}\hline & 1 & \\ \hline 1 & -12 & 1 \\ \hline & 1 & \\ \hline \end{array}\,D_{i,j} + \frac{1}{6\,h^2}\begin{array}{|c|c|c|}\hline 1 & & 1 \\ \hline & -4 & \\ \hline 1 & & 1 \\ \hline \end{array}\,\Psi_{i,j}. \tag{3.19}$$

As in the one-dimensional case, both formulations of the 2SHOC schemes in two dimensions are computationally equivalent to the non-compact fourth-order stencil

$$\nabla^2\Psi_{i,j} = -\frac{1}{12\,h^2}$$

		1		
		-16		
1	-16	60	-16	1
		-16		
		1		

$$\Psi_{i,j} + O(h^4). \tag{3.20}$$

3.2.3 Three-Dimensional Formulation

In three dimensions, the Laplacian operator applied to Ψ can be represented as

$$\begin{aligned}\nabla^2\Psi_{i,j,k} &= \left.\frac{\partial^2\Psi}{\partial x^2}\right|_{i,j,k} + \left.\frac{\partial^2\Psi}{\partial y^2}\right|_{i,j,k} + \left.\frac{\partial^2\Psi}{\partial z^2}\right|_{i,j,k} \\ &= \delta_x^2\Psi_{i,j,k} + \delta_y^2\Psi_{i,j,k} + \delta_z^2\Psi_{i,j,k} \\ &\quad - \frac{h^2}{12}\left(\left.\frac{\partial^4\Psi}{\partial x^4}\right|_{i,j,k} + \left.\frac{\partial^4\Psi}{\partial y^4}\right|_{i,j,k} + \left.\frac{\partial^4\Psi}{\partial y^4}\right|_{i,j,k}\right) + O(h^4).\end{aligned} \tag{3.21}$$

There are again two formulations of the 2SHOC scheme which trade off storage requirement versus number of computations. The first formulation takes *three* storage arrays (D^x, D^y, D^z), while, as in the two-dimensional case, the other formulation takes only *one* (D). The three-storage 2SHOC directly follows from the two-dimensional

version and is defined as

$$1) \qquad D^x_{i,j,k} = \delta^2_x \Psi_{i,j,k} = \frac{\Psi_{i+1,j,k} - 2\Psi_{i,j,k} + \Psi_{i-1,j,k}}{h^2} \tag{3.22}$$
$$D^y_{i,j,k} = \delta^2_y \Psi_{i,j,k} = \frac{\Psi_{i,j+1,k} - 2\Psi_{i,j,k} + \Psi_{i,j-1,k}}{h^2}$$
$$D^z_{i,j,k} = \delta^2_z \Psi_{i,j,k} = \frac{\Psi_{i,j,k+1} - 2\Psi_{i,j,k} + \Psi_{i,j,k-1}}{h^2}$$

$$2) \qquad \nabla^2 \Psi_{i,j,k} \approx \frac{7}{6}\left(D^x_{i,j,k} + D^y_{i,j,k} + D^z_{i,j,k}\right) \tag{3.23}$$
$$- \frac{1}{12}\left(D^x_{i+1,j,k} + D^x_{i-1,j,k} + D^y_{i,j+1,k} + D^y_{i,j-1,k} + D^z_{i,j,k+1} + D^z_{i,j,k-1}\right).$$

The single storage formulation requires the use of the cross-derivatives of $\nabla^2(\nabla^2\Psi)$ as before. We write the fourth derivative truncation terms of Eq. (3.21) as

$$\frac{\partial^4 \Psi}{\partial x^4} + \frac{\partial^4 \Psi}{\partial y^4} + \frac{\partial^4 \Psi}{\partial z^4} = \nabla^2\left(\nabla^2 \Psi\right) - 2\left(\frac{\partial^4 \Psi}{\partial x^2\, \partial y^2} + \frac{\partial^4 \Psi}{\partial x^2\, \partial z^2} + \frac{\partial^4 \Psi}{\partial y^2\, \partial z^2}\right), \tag{3.24}$$

therefore the cross-derivatives in Eq. (3.24) can be approximated to second-order accuracy using nine-point compact stencils in each direction which contain the same coefficients as the two-dimensional stencil of Eq. (3.16). We can now write a second-order accurate approximation of Eq. (3.24) as

$$\nabla^2 \Psi_{i,j,k} \approx \tag{3.25}$$
$$\frac{1}{h^2}\left(\begin{array}{|c|c|c|} \hline & & \\ \hline & 1 & \\ \hline & & \\ \hline \end{array} D_{i,j+1,k} + \begin{array}{|c|c|c|} \hline & 1 & \\ \hline 1 & -6 & 1 \\ \hline & 1 & \\ \hline \end{array} D_{i,j,k} + \begin{array}{|c|c|c|} \hline & & \\ \hline & 1 & \\ \hline & & \\ \hline \end{array} D_{i,j-1,k} \right)$$
$$-\frac{2}{h^4}\left(\begin{array}{|c|c|c|} \hline & 1 & \\ \hline 1 & -4 & 1 \\ \hline & 1 & \\ \hline \end{array} \Psi_{i,j+1,k} + \begin{array}{|c|c|c|} \hline 1 & -4 & 1 \\ \hline -4 & 12 & -4 \\ \hline 1 & -4 & 1 \\ \hline \end{array} \Psi_{i,j,k} + \begin{array}{|c|c|c|} \hline & 1 & \\ \hline 1 & -4 & 1 \\ \hline & 1 & \\ \hline \end{array} \Psi_{i,j-1,k} \right),$$

where $D_{i,j,k} = \delta_x^2 \Psi_{i,j,k} + \delta_y^2 \Psi_{i,j,k} + \delta_z^2 \Psi_{i,j,k}$ is the second-order approximation to the Laplacian. Once again, part of the Ψ stencil of Eq. (3.25) can be written in terms of D, in which case the single-storage 2SHOC scheme after simplification is given as

$$1)\ D_{i,j,k} = \tag{3.26}$$

$$\frac{1}{h^2}\left(\begin{array}{|c|c|c|}\hline & & \\ \hline & 1 & \\ \hline & & \\ \hline \end{array} \Psi_{i,j+1,k} + \begin{array}{|c|c|c|}\hline & 1 & \\ \hline 1 & -6 & 1 \\ \hline & 1 & \\ \hline \end{array} \Psi_{i,j,k} + \begin{array}{|c|c|c|}\hline & & \\ \hline & 1 & \\ \hline & & \\ \hline \end{array} \Psi_{i,j-1,k} \right),$$

$$2)\ \nabla^2 \Psi_{i,j,k} \approx \tag{3.27}$$

$$-\frac{1}{12}\left(\begin{array}{|c|c|c|}\hline & & \\ \hline & 1 & \\ \hline & & \\ \hline \end{array} D_{i,j+1,k} + \begin{array}{|c|c|c|}\hline & 1 & \\ \hline 1 & -10 & 1 \\ \hline & 1 & \\ \hline \end{array} D_{i,j,k} + \begin{array}{|c|c|c|}\hline & & \\ \hline & 1 & \\ \hline & & \\ \hline \end{array} D_{i,j-1,k} \right)$$

$$+\frac{1}{6\,h^2}\left(\begin{array}{|c|c|c|}\hline & 1 & \\ \hline 1 & & 1 \\ \hline & 1 & \\ \hline \end{array} \Psi_{i,j+1,k} + \begin{array}{|c|c|c|}\hline 1 & & 1 \\ \hline & -12 & \\ \hline 1 & & 1 \\ \hline \end{array} \Psi_{i,j,k} + \begin{array}{|c|c|c|}\hline & 1 & \\ \hline 1 & & 1 \\ \hline & 1 & \\ \hline \end{array} \Psi_{i,j-1,k} \right).$$

Again, both formulations of the 2SHOC scheme in three dimensions are computationally equivalent to the standard non-compact fourth-order stencil.

3.3 NUMERICAL RESULTS

Here we show our numerical results for using the RK4+2SHOC scheme to integrate the NLSE. Since the 2SHOC schemes are algebraically equivalent to the standard fourth-order non-compact schemes, they should exhibit the same order of accuracy as well. However, because the order of computation is altered (which could introduce numeric cancellation or round-off errors), and the implementation of boundary conditions is different than when using wide stencils, numerical tests of the accuracy of

the 2SHOC scheme is justified. For all numerical tests we use our GPU-accelerated code package NLSEmagic (see Chapters 7 and 8) to integrate the NLSE and pick non-trivial initial conditions. For all tests we utilize the single-storage formulation of the 2SHOC schemes where applicable.

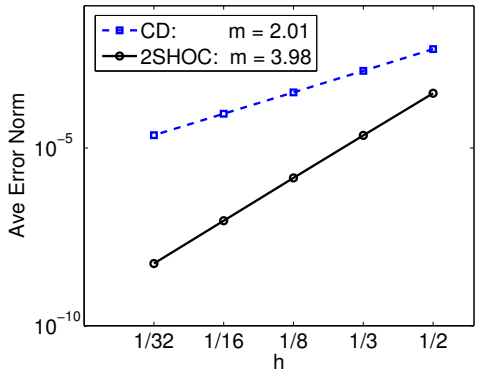
To compute the error in the simulations, the real and imaginary parts of the wavefunction Ψ are compared to the true solution at 100 equal-spaced intervals throughout the simulation. The averaged L2-norm of the wavefunction error ($E^n = \|\Psi^n - \Psi^n_{\text{exact}}\|_2/\mathbf{N}$, where n is the current time step and $\mathbf{N}$ is the total number of grid points) is computed at each time interval. Using the averaged L2-norm error is necessary since we are using a fixed domain and therefore the total L2-norm would be greater for smaller h due to the increase in the number of grid points. When the simulation is completed, we define the error of the real and imaginary parts of the whole simulation as the mean of the errors at each of the 100 intervals ($E = \Sigma E^n/K$, where $K = 100$ is the number of time intervals). To compute the order of error, we define the error order between two simulations with spatial steps h_1 and h_2 as $O = \ln(E_{h_1}) - \ln(E_{h_2})$. The overall order of the scheme is determined by taking the mean of the average of the real and imaginary error orders for each run. For comparison, we run each simulation using the classic second-order central-differencing (CD) in space as well.

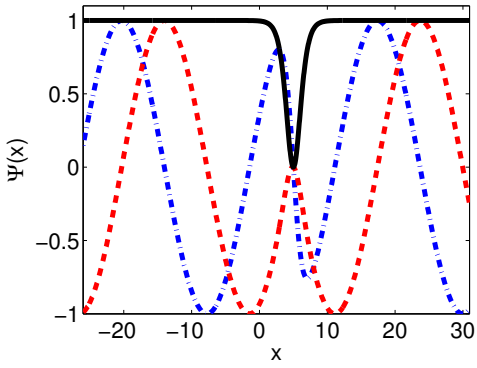
We note that since the RK4 scheme is $O(\mathbf{k}^4)$, and the stability bounds require that $\mathbf{k} \propto h^2$, the errors in the simulations attributed to the time-stepping are negligible compared to those due to the spatial differencing, and therefore should not effect the accuracy tests.

3.3.1 One-Dimensional Test

For the one-dimensional test of the RK4+2SHOC scheme, we use the exact solution to the one-dimensional NLSE (with $V(\mathbf{r}) = \mathbf{0}$ and $s < 0$) of a co-moving dark soliton given by Eq. (2.5) in Chapter 2. We use the modulus-squared Dirichlet boundary condition ($|\Psi|^2 =$ constant) developed in Chapter 4 and set the size of the computational

domain large enough so that the wavefunction's density is machine epsilon ($\epsilon \approx 10^{-16}$) lower than the background density at the boundaries throughout the entire time of the simulation. We set $s = -1$, $a = 1$, $c = 0.5$, and $\Omega = -1$ in Eq. (2.5). The grid spatial-step h is varied from $1/2$ to $1/32$. We run the simulation to an end-time of $t = 10$ with a time-step size of $\mathbf{k} = 0.0005$ (which is slightly less than the maximum stability bound [see Chapter 5] for the smallest value of h used) resulting in $20,000$ time steps. The results of the simulations are shown in Fig. 3.1, where the fourth-order accuracy of the RK4+2SHOC scheme is easily observed.





	$h = 1/2$	$h = 1/4$	$h = 1/8$	$h = 1/16$	$h = 1/32$
CD:					
Error (real)	0.0060534	0.0014769	0.0003673	9.1733e-05	2.293e-05
Order (real)	–	2.0352	2.0075	2.0015	2.0002
Error (imag)	0.0060159	0.0014675	0.00036496	9.1146e-05	2.2784e-05
Order (imag)	–	2.0354	2.0076	2.0015	2.0002
2SHOC:					
Error (real)	0.00034675	2.2662e-05	1.4374e-06	9.0208e-08	5.6444e-09
Order (real)	–	3.9355	3.9788	3.9941	3.9984
Error (imag)	0.00034481	2.2538e-05	1.4297e-06	8.9723e-08	5.6141e-09
Order (imag)	–	3.9353	3.9786	3.994	3.9984

Figure 3.1. Top Left: Overall order of accuracy (m) computed from simulating the one-dimensional NLSE with the initial condition of Eq. (2.5) for the RK4+CD and RK4+2SHOC schemes. Top Right: Snap-shot of the simulation with $h = 1/32$. The solid (black) line is the modulus-squared $|\Psi|^2$, while the dot-dashed (blue) and dashed (red) lines are the real and imaginary parts of Ψ respectively. Bottom: Table of error values and order of accuracy values for each spatial step h. The parameters used for the solution are $s = -1$, $a = 1$, $c = 0.5$, and $\Omega = -1$. The solution is integrated to an end-time of $t = 10$ with a time-step size of $\mathbf{k} = 0.0005$.

3.3.2 Two-Dimensional Test

There is no readily available, non-trivial, exact two-dimensional solution to the NLSE to use for order comparisons. However, since the RK4+2SHOC scheme is explicit (and therefore does not require any special handling of the nonlinearity such as iterative methods), the accuracy of the scheme can be tested reliably in a linear setting (where $s = 0$). The chosen test problem is the Gaussian wave-packet solution

$$\Psi(x, y, t) = \exp\left(-\frac{x^2 + y^2}{2\,a}\right) \exp(-i\,2\,t), \tag{3.28}$$

which is an exact solution to the NLSE when the external potential $V(x, y)$ is defined as

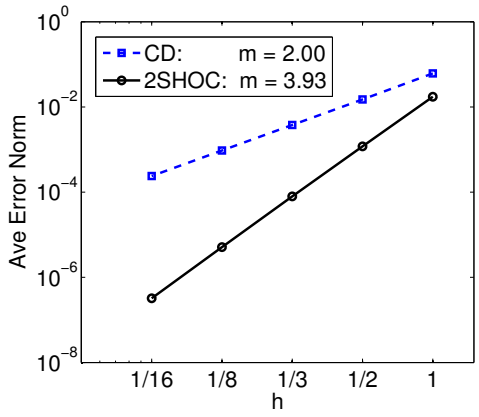
$$V(x, y) = \frac{x^2 + y^2}{a}. \tag{3.29}$$

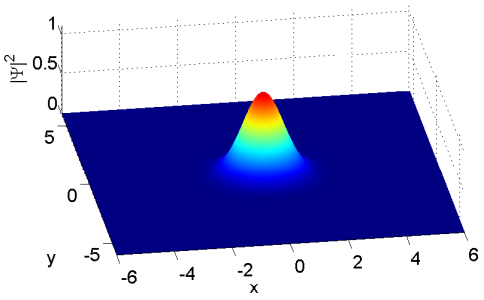
We use Dirichlet boundary conditions ($\Psi = 0$) and set $a = 1$. The size of the computational domain is set to be large enough so that the solution has a value of $\Psi_b = \sqrt{\epsilon}$ (where $\epsilon \approx 10^{-16}$) at the boundaries. The simulation time and number of intervals for error computations are the same as in the one-dimensional test in Sec. 3.3.1. We vary h from $h = 1$ to $h = 1/16$ and use a time-step of $\mathbf{k} = 0.001$ resulting in $10,000$ time steps. The results are shown in Fig. 3.2. Like in the one-dimensional case, the fourth-order accuracy of the RK4+2SHOC scheme is observed.

3.3.3 Three-Dimensional Test

As in the two-dimensional case, there is no readily available non-trivial exact solution to test the three-dimensional NLSE. Therefore, we again use a linear example choosing the three-dimensional analog of Eq. (3.28) defined as

$$\Psi(x, y, t) = \exp\left(-\frac{x^2 + y^2 + z^2}{2\,a}\right) \exp(-i\,3\,t), \tag{3.30}$$





	$h = 1$	$h = 1/2$	$h = 1/4$	$h = 1/8$	$h = 1/16$
CD:					
Error (real)	0.059782	0.01451	0.0036626	0.00092345	0.00023196
Order (real)	–	2.0426	1.9861	1.9878	1.9932
Error (imag)	0.062855	0.015442	0.0038944	0.00097946	0.00024585
Order (imag)	–	2.0252	1.9874	1.9913	1.9942
2SHOC:					
Error (real)	0.017015	0.0011492	7.7072e-005	4.9388e-006	3.1147e-007
Order (real)	–	3.8882	3.8982	3.964	3.987
Error (imag)	0.017684	0.001228	8.2419e-005	5.2815e-006	3.3308e-007
Order (imag)	–	3.848	3.8972	3.964	3.987

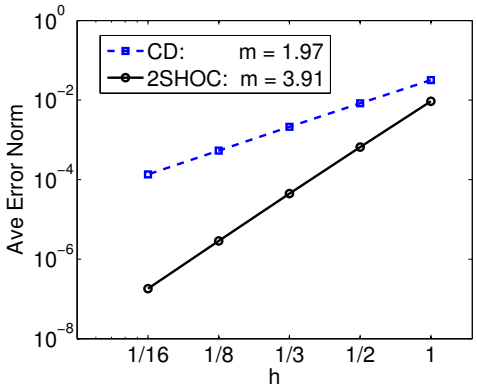
Figure 3.2. Top Left: Overall order of accuracy (m) computed from simulating the two-dimensional linear Schrödinger equation with the initial condition of Eq. (3.28) for the RK4+CD and RK4+2SHOC schemes. Top Right: Depiction of the modulus-squared $|\Psi|^2$ of the wavefunction with $h = 1/16$ at $t = 0$. Bottom: Table of error values and order of accuracy values for each spatial step h. The parameters used for the solution are $s = 0$, and $a = 1$. The solution is integrated to an end time of $t = 10$ with a time-step size of $\mathrm{k} = 0.001$.

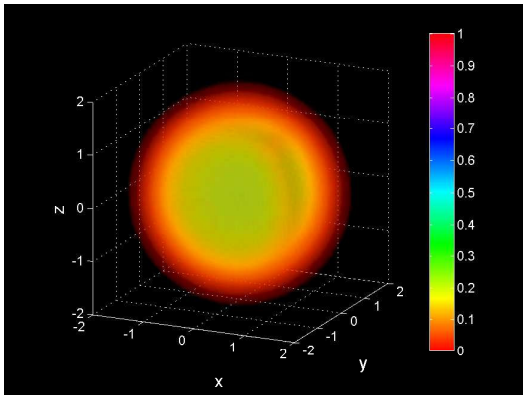
with the external potential

$$V(x, y, z) = \frac{x^2 + y^2 + z^2}{a}. \tag{3.31}$$

As before, Dirichlet boundary conditions ($\Psi = 0$) are used and a is set to 1. The size of the computational domain is once again set to be large enough so that the exact solution has a value of $\Psi_b = \sqrt{\epsilon}$ at the boundaries. The simulation time and number of intervals for error computations are the same as in the one-dimensional test in Sec. 3.3.1. The step-size is varied from $h = 1$ to $h = 1/16$ and a time-step of $\mathbf{k} = 0.0005$ is used resulting in $20,000$ time steps. The results are shown in Fig. 3.3, where once again the

fourth-order accuracy of the scheme is clearly seen. The average order of accuracy given





	$h = 1$	$h = 1/2$	$h = 1/4$	$h = 1/8$	$h = 1/16$
CD:					
Error (real)	0.03125	0.0083706	0.0021518	0.00054571	0.00013743
Order (real)	–	1.9004	1.9598	1.9793	1.9894
Error (imag)	0.032541	0.0083009	0.0021026	0.00053144	0.00013374
Order (imag)	–	1.9709	1.9811	1.9842	1.9905
2SHOC:					
Error (real)	0.0093495	0.000666	4.5165e-005	2.9087e-006	1.839e-007
Order (real)	–	3.8113	3.8822	3.9567	3.9834
Error (imag)	0.0093489	0.00064941	4.4012e-005	2.8344e-006	1.792e-007
Order (imag)	–	3.8476	3.8832	3.9568	3.9834

Figure 3.3. Top Left: Overall order of accuracy (m) computed from simulating the three-dimensional linear Schrödinger equation with the initial condition of Eq. (3.30) for the RK4+CD and RK4+2SHOC schemes. Top Right: Volumetric rendering of the modulus-squared $|\Psi|^2$ of the wavefunction with $h = 1/16$ at $t = 0$ (displayed on a smaller grid than used in the simulations). Bottom: Table of error values and order of accuracy values for each spatial step h. The parameters used for the solution are $s = 0$, and $a = 1$. The solution is integrated to an end time of $t = 10$ with a time-step size of $\mathbf{k} = 0.0005$.

in Fig. 3.3 is 3.91 which is slightly smaller than expected for the fourth-order scheme. However as seen in the corresponding table, the order of accuracy starts at around 3.8 for the first spatial step-size reduction, while in the smaller reductions, the order increases to around 3.98. This means that the slightly lower-order values are most likely due to the inability of the coarse grid to adequately resolve the solution, and not on the scheme itself. Lower values of h were not used due to computational memory constraints.

From the table in Fig. 3.3, it is easy to illustrate the advantage of using high-order schemes for large three-dimensional problems mentioned in the beginning of this chapter — that of being able to use a much smaller grid while maintaining the desired accuracy. For example, in Fig. 3.3, using the second-order scheme for $h = 1/8$ (making the grid resolution $97 \times 97 \times 97 = 912,673$ grid points) yielded an error of around 0.0005. Roughly the same error (0.0006) was found using the fourth-order scheme with a spatial-step size of $h = 1/2$, requiring the grid size of only $25 \times 25 \times 25 = 15,625$ grid points. This is a 98.3% reduction in the number of grid points required! When combined with the ease of parallelization that the 2SHOC compact scheme provides, its usefulness for large three-dimensional problems is apparent.

3.4 Computation and Storage Comparisons

In this section we compare the storage and computational requirements of the 2SHOC schemes compared to the standard fourth-order non-compact equivalent schemes for one, two and three dimensions. We count the number of operations (in terms of the number of total grid points in the domain, $\mathbf{N}$) needed for each method treating the boundaries as internal points. For simplicity, the added operations needed due to Ψ being complex are ignored treating all the variables as real. Also, since floating-point division operations are far more computationally expensive than additions and multiplications, we record the operations required in an optimized form of the schemes, in which all divisions are only pre-computed once. We record the number of computations and storage space required for the Laplacian operator alone using the 2SHOC schemes, as well as when implemented into the NLSE simulations using the RK4+2SHOC schemes.

The one-dimensional analysis is shown in Table 3.1. We see that for the RK4+2SHOC NLSE implementation, the scheme only requires about 6% more computations and 20% more storage than the equivalent non-compact scheme. This small increase in storage and computations is minor compared to the advantages of having a compact scheme.

Table 3.1. One-dimensional storage and computational cost analysis for the 2SHOC scheme compared to the equivalent non-compact scheme. The storage is given in terms of N, the total number of grid points.

Method	Operations	Storage	Op Ratio	Storage Ratio
Laplacian:				
Non-Compact	7	N	–	–
2SHOC	8	2N	1.14	2
NLSE RK4 Step:				
Non-Compact	4 (7+7)+13 = 69	5N	–	–
2SHOC	4 (8+7)+13 = 73	6N	1.06	1.2

The two-dimensional analysis is given in Table 3.2. Both the double-storage version of the 2SHOC of Eqs. (3.13) and (3.14) and the single-storage version of Eqs. (3.18) and (3.19) are analyzed. Here, using the 2SHOC scheme has a greater increase

Table 3.2. Two-dimensional storage and computational cost analysis for 2SHOC schemes compared to equivalent non-compact schemes. The storage is given in terms of N, the total number of grid points.

Method	Operations	Storage	Op Ratio	Storage Ratio
Laplacian:				
Non-Compact	9	N	–	–
2SHOC (2X Storage)	15	3N	1.66	3
2SHOC (1X Storage)	19	2N	2.11	2
NLSE RK4 Step:				
Non-Compact	4 (9+7) +13 = 77	5N	–	–
2SHOC (2X Storage)	4 (15+7)+13 = 101	7N	1.31	1.4
2SHOC (1X Storage)	4 (19+7)+13 = 117	6N	1.52	1.2

in additional computations, taking approximately 50% more (for the single (1X) storage version) than the non-compact scheme in the RK4 step of the NLSE. The double (2X) storage version of 2SHOC reduces this to about 30%, but with a 20% increase in the storage required when compared to the single-storage 2SHOC (which, like the one-dimensional case, is only 20% more than the non-compact scheme).

The three-dimensional analysis is given in Table 3.3. Here we see that the single-storage 2SHOC scheme in the RK4 step for the NLSE takes 70% more

Table 3.3. Three-dimensional storage and computational cost analysis for 2SHOC schemes compared to equivalent non-compact schemes. The storage is given in terms of N, the total number of grid points.

Method	Operations	Storage	Op Ratio	Storage Ratio
Laplacian:				
Non-Compact	14	$\mathbf{N}$	–	–
2SHOC (3X Storage)	22	$4\mathbf{N}$	1.57	4
2SHOC (1X Storage)	31	$2\mathbf{N}$	2.21	2
NLSE RK4 Step:				
Non-Compact	4 (14+7)+13 = 97	$5\mathbf{N}$	–	–
2SHOC (3X Storage)	4 (22+7)+13 = 129	$8\mathbf{N}$	1.33	1.6
2SHOC (1X Storage)	4 (31+7)+13 = 165	$6\mathbf{N}$	1.70	1.2

computations than the non-compact scheme, and for the Laplacian operator alone, takes a little more than twice the computations required for the non-compact scheme. The 1X-storage 2SHOC scheme for the Laplacian only takes 57% more computations but also uses four times the storage. Since the main advantage of using a compact scheme is that it is easy to implement in a parallel environment and to realize near the boundaries, the increase in computation is easily offset by the parallelism (as most parallel implementations have speed-ups of well over two). In addition, often times, the number of computations an algorithm takes is not nearly as important as the memory bandwidth used. Therefore, using less memory but more computations may actually perform better than using less computations and more memory. For this reason, we choose to use the single-storage 2SHOC schemes in the codes to be presented in Chapters 7 and 8. Also, since memory capacity can be limited in certain computational environments such as GPUs, using the single-storage 2SHOC schemes allow for larger problem sizes.

In the discussion of the RK4-2SHOC schemes so far, we have left out the details of implementing boundary conditions to the schemes, a topic we now discuss in the next chapter.

CHAPTER 4

NUMERICAL ALGORITHMS: MODULUS-SQUARED DIRICHLET BOUNDARY CONDITION

As described in Sec. 1.1.4, for the simulations to be performed in the current study, a constant modulus-square boundary condition is desired. In this chapter, we present a simple way to simulate a modulus-squared Dirichlet boundary condition (MSD) for the NLSE which also can be used for *any* time-dependent complex-valued PDE. The MSD boundary condition is very easy to implement and has an accuracy equal to the interior numerical scheme (as long as the assumption of a constant modulus-squared value at the boundary is valid). This new boundary condition eliminates the need for overly large grids or expensive and complicated boundary conditions for many problems, including the NLSE.

A common boundary technique to approximate an infinite-domain is to use periodic boundary conditions where the boundary cell is treated as if the opposite boundary cell is adjacent to it. For bright structures in the NLSE, periodic boundary conditions work well since the coherent structures decay to zero rapidly and therefore will be unaffected by a 'repeated' copy of itself a domain length away. However, in constant-density background scenarios, a coherent dark structure instantaneously 'feels' any other structure or phase gradient in the domain and therefore using a periodic boundary condition can cause dynamical effects on dark structures which would not happen in a true infinite background. Additionally, for some solutions (like the steady-state dark soliton in Eq. (2.4) in Chapter 2) a large phase jump can exist between one boundary of the domain and the opposite boundary. Using periodic boundary

conditions artificially connects this phase jump at the boundaries and the resulting solution is not the original one desired (isolated in an infinite background) [101]. Since the main focus in this dissertation is dark coherent structures on an infinite background density domain, periodic boundary conditions are not utilized.

4.1 FORMULATION OF THE MSD BOUNDARY CONDITION

To formulate a MSD boundary condition we first introduce the notation that Ψ_b describes the grid-points on a boundary and Ψ_{b-1} describes the grid-points one cell in from the boundary in the normal direction to the boundary points (i.e. in two-dimensions, the top-left corner point would be a Ψ_b and the point one cell to the right and and one cell down would be Ψ_{b-1}). The real part of Ψ is denoted as Ψ^R and the imaginary part as Ψ^I. The short-hand notation of Ψ_α is used to denote the first derivative with respect to α ($\partial\Psi/\partial\alpha$). So, for example,$\Psi^R_{\alpha,b}$ would refer to the real-part of the first partial derivative with respect to α at the boundary points, where α can represent a spatial (x) or temporal (t) dependence.

We start out by stating the condition, that for all times, the modulus-squared of the function Ψ at the boundaries is equal to a constant, positive, real value B,

$$|\Psi_b|^2 = B. \tag{4.1}$$

In such a case we have that

$$\frac{\partial}{\partial t}\,|\Psi_b|^2 = 0,$$

and thus

$$\frac{\partial}{\partial t}\,\left(\Psi^R_b\right)^2 + \frac{\partial}{\partial t}\,\left(\Psi^I_b\right)^2 = 0. \tag{4.2}$$

By using the chain rule and rearranging, Eq. (4.2) is equivalent to

$$\Psi^R_b\,\frac{\partial \Psi^R_b}{\partial t} = -\Psi^I_b\,\frac{\partial \Psi^I_b}{\partial t}. \tag{4.3}$$

A non-trivial solution that satisfies Eq. (4.3) is given by

$$\frac{\partial \Psi_b^R}{\partial t} = C\,\Psi_b^I \qquad \text{and} \qquad \frac{\partial \Psi_b^I}{\partial t} = -C\,\Psi_b^R, \tag{4.4}$$

where C is a constant. Since $\Psi_{t,b} = \Psi_{b,t}^R + i\,\Psi_{b,t}^I$, Eq. (4.4) can be expressed as

$$\Psi_b = \frac{i}{C}\,\Psi_{t,b}. \tag{4.5}$$

Eq. (4.5) gives the form of the MSD boundary condition on Ψ_t in terms of the boundary point Ψ_b and the constant C. In order to find the correct value of C, we differentiate Eq. (4.5) in the normal direction to the boundary (here represented as the x-direction) to get

$$\frac{\partial}{\partial x}\Psi_b = \frac{i}{C}\,\frac{\partial}{\partial x}\Psi_{t,b}. \tag{4.6}$$

Discretizing the spatial derivatives of Eq. (4.6) using forward-differencing yields

$$\frac{\Psi_{b-1} - \Psi_b}{h} - \frac{h}{2}\Psi_{xx,b} + O(h^2) = \frac{i}{C}\left[\frac{\Psi_{t,b-1} - \Psi_{t,b}}{h} - \frac{h}{2}\Psi_{txx,b} + O(h^2)\right], \tag{4.7}$$

where h is the grid spacing between the boundary and the interior point in the normal direction. Inserting Eq. (4.5) into Eq. (4.7) to eliminate temporal derivatives at the boundary points yields

$$\frac{\Psi_{b-1} - \Psi_b}{h} - \frac{h}{2}\Psi_{xx,b} + O(h^2) = \frac{i}{C}\left[\frac{\Psi_{t,b-1} + i\,C\Psi_b}{h} + \frac{i\,h\,C}{2}\Psi_{xx,b} + O(h^2)\right]. \tag{4.8}$$

It is observed that the first truncation terms shown in Eq. (4.8) cancel out, and by the same formulation, all the additional truncation terms will also cancel out. Therefore, we have

$$\frac{\Psi_{b-1} - \Psi_b}{h} = \frac{i}{C}\left[\frac{\Psi_{t,b-1} + i\,C\Psi_b}{h}\right], \tag{4.9}$$

which yields the expression for C given by

$$C = i\,\frac{\Psi_{t,b-1}}{\Psi_{b-1}}. \tag{4.10}$$

Eq. (4.10) is known since the value of $\Psi_{t,b-1}$ is computed using the interior scheme being implemented. Inserting the value of C of Eq. (4.10) into the MSD boundary condition formulation of Eq. (4.5) yields

$$\Psi_{t,b} \approx \frac{\Psi_{t,b-1}}{\Psi_{b-1}}\,\Psi_b. \tag{4.11}$$

We note that the MSD boundary condition of Eq. (4.11) is very similar in form to the Sommerfeld radiation boundary condition mentioned in Ref. [70] where the time-derivative at the boundary is computed as $\Psi_{t,b} \approx -(\Psi_{t,b-1}/\Psi_{n_r,b-1})\Psi_{n_r,b}$ where n_r is the direction normal to the boundary. As mentioned in Chapter 1, the radiation boundary condition of Ref. [70] has been used successfully in the simulations of dark vortex rings in the NLSE [32], however using it requires additional computational steps and storage as it requires the computation of spacial derivative terms which are not normally calculated in the NLSE simulations on all points adjacent to the boundaries. Since storage and efficiency are very important for large simulations (especially on GPU cards, see Chapter 7), the radiation boundary condition of Ref. [70] is not implemented in our codes.

The MSD boundary condition of Eq. (4.11) can be obtained in an alternative way by noting that a steady-state (in the modulus-squared) of a complex time-dependent function can be given as

$$\Psi = |\Psi| \exp(-i\,\Omega\,t),$$

where Ω is the frequency and $|\Psi|$ is the time-independent modulus of Ψ. If we assume that this relationship is true at the boundary points of the grid, we have

$$\frac{\partial \Psi_b}{\partial t} = -i\,\Omega_b\,|\Psi_b| \exp(i\,\Omega_b\,t) = -i\,\Omega_b\,\Psi_b. \tag{4.12}$$

From Eq. (4.12), we get

$$\Omega_b = i\,\frac{\Psi_{t,b}}{\Psi_b}. \tag{4.13}$$

Assuming that the closest interior grid point is nearly steady-state, we can infer that

$$\Omega_{b-1} \approx i\,\frac{\Psi_{t,b-1}}{\Psi_{b-1}}, \tag{4.14}$$

and that the frequency of the interior point is approximately equal to the frequency at the boundary (i.e. $\Omega_b \approx \Omega_{b-1}$). Then, by inserting Eq. (4.14) as Ω_b in Eq. (4.12), Eq. (4.12) becomes the MSD boundary condition of Eq. (4.11).

This formulation of the MSD boundary condition shows that the $\Psi_{t,b}/\Psi_b$ term in Eq. (4.11) should be equal to i times the frequency. Since the frequency of the solution is a real value, computing $\Psi_{t,b-1}/\Psi_{b-1}$ may introduce a small imaginary part to the frequency due to numerical errors. This would cause the solution at the boundaries to undergo eventual exponential growth (which we have observed in simulations not reported here). In order to ensure that the computed frequency is real-valued (i.e. that $\Psi_{t,b-1}/\Psi_{b-1}$ is purely imaginary), we modify the MSD boundary condition of Eq. (4.11) to be

$$\Psi_{t,b} \approx i\,\mathrm{Im}\left[\frac{\Psi_{t,b-1}}{\Psi_{b-1}}\right]\,\Psi_b. \tag{4.15}$$

When using the MSD boundary condition in programming environments that do not intrinsically handle complex variables, Eq. (4.15) must be explicitly split into its real and imaginary parts. We begin by expanding the unmodified MSD boundary condition of Eq. (4.11) into its real and imaginary parts yielding

$$\Psi^R_{t,b} + i\,\Psi^I_{t,b} \approx \frac{\Psi^R_{t,b-1} + i\,\Psi^I_{t,b-1}}{\Psi^R_{b-1} + i\,\Psi^I_{b-1}}\left(\Psi^R_b + i\,\Psi^I_b\right).$$

which leads to

$$\Psi_{t,b}^{R} + i\,\Psi_{t,b}^{I} \approx \left[\frac{\Psi_{t,b-1}^{R}\,\Psi_{b-1}^{R} + \Psi_{t,b-1}^{I}\,\Psi_{b-1}^{I}}{\left(\Psi_{b-1}^{R}\right)^{2} + \left(\Psi_{b-1}^{I}\right)^{2}} + i \left(\frac{\Psi_{t,b-1}^{I}\,\Psi_{b-1}^{R} - \Psi_{t,b-1}^{R}\,\Psi_{b-1}^{I}}{\left(\Psi_{b-1}^{R}\right)^{2} + \left(\Psi_{b-1}^{I}\right)^{2}} \right) \right] \left(\Psi_{b}^{R} + i\,\Psi_{b}^{I}\right). \tag{4.16}$$

Going back to Eq. (4.3), we see that the first term in the brackets of Eq. (4.16) is equal to zero at the boundary. If it is assumed that the interior points are similar, the term can be dropped. Dropping the term is equivalent to the numerical fix used in Eq. (4.15), ensuring that the computed frequency of the solution Ψ at the boundary is real-valued. Simplifying Eq. (4.16) with this in mind yields the separated MSD boundary condition

$$\begin{aligned} \Psi_{t,b}^{R} &= -\tilde{\Omega}\,\Psi_{b}^{I} \\ \Psi_{t,b}^{I} &= \tilde{\Omega}\,\Psi_{b}^{R}, \end{aligned} \tag{4.17}$$

where

$$\tilde{\Omega} = \frac{\Psi_{t,b-1}^{I}\,\Psi_{b-1}^{R} - \Psi_{t,b-1}^{R}\,\Psi_{b-1}^{I}}{\left(\Psi_{b-1}^{R}\right)^{2} + \left(\Psi_{b-1}^{I}\right)^{2}}. \tag{4.18}$$

The MSD boundary condition of Eqs. (4.15), (4.17), and (4.18) is given for the temporal derivative of the boundary point. This is ideally suited for Runge-Kutta time-stepping schemes such as the RK4 described in Chapter 3, as the right hand side of the PDEs (Ψ_t) are evaluated and used for the time-stepping. For other methodologies, or in situations where the boundary value of the spatial derivatives is needed, the MSD boundary condition can be inserted into the governing equation to formulate the required values. An example of this is shown in Sec. 4.2 for the NLSE.

In situations where the sequential computation of the internal scheme followed by the boundary condition computations is not appropriate (for example, implicit finite-difference schemes) one can substitute the internal scheme of $\Psi_{t,b-1}$ into Eq. (4.15) and implement the boundary condition concurrently with the internal scheme (this would

add extra computational steps as $\Psi_{t,b-1}$ would essentially be computed twice). Alternatively, if the frequency, Ω, of the overall solution is exactly known, the $\Psi_{t,b-1}/\Psi_{b-1}$ term can be replaced with $i\,\Omega$ directly as shown above.

If it happens that $\Psi_{b-1} = 0$, the MSD boundary condition of Eq. (4.11) has a singularity. However, in most situations, Ψ_{b-1} only takes a zero value when the solution is tending toward zero at the boundary, in which case, the standard Dirichlet boundary condition of $\Psi_b = 0$ can be used instead of the MSD boundary condition. Alternatively, one can numerically check the values of Ψ_{b-1}, and use an alternative boundary condition in the case that $\Psi_b = 0$.

The following are a few key features of the MSD boundary condition:

- The standard time-derivative formulation of the MSD boundary condition does not depend at all on the specific PDE being simulated, only that it is time-dependent and complex-valued.
- The MSD boundary condition can be used in multidimensional settings without modification since each boundary point only uses one interior point in the normal direction to the boundary.
- The MSD boundary condition does not depend on the size of the grid spacing, and therefore does not need to be altered when using unequal grid spacing (for example, in adaptive mesh refinement applications).
- The MSD boundary condition can be considered compact, in that after computing the internal scheme, the boundary values only depend on their nearest neighboring grid point. This allows the MSD boundary condition to be easily implemented in parallel environments.
- In general, the MSD boundary condition is extremely easy to implement (see Appendix E for MATLAB code showing an implementation of the MSD boundary condition in one-dimension). This makes the MSD boundary condition an attractive alternative to more complicated boundary condition methodologies.

4.2 Application of the MSD Boundary Condition to the NLSE

As mentioned in Sec. 4.1, since the MSD boundary condition of Eq. (4.15) is given in terms of the temporal derivative, it is well suited for Runge-Kutta schemes, in which case

it can be applied directly. However, other numerical schemes require boundary conditions on the Laplacian operator itself. In the case of the NLSE, this can be worked out by inserting the NLSE of Eq. (1.2) into Eq. (4.15) yielding

$$\nabla^2\Psi_b \approx \left[\mathrm{Im}\left(i\,\frac{\nabla^2\Psi_{b-1}}{\Psi_{b-1}}\right) + \frac{1}{a}\,\left(N_{b-1} - N_b\right)\right]\Psi_b, \tag{4.19}$$

where

$$N_b = s\,|\Psi_b|^2 - V_b, \qquad N_{b-1} = s\,|\Psi_{b-1}|^2 - V_{b-1}. \tag{4.20}$$

Splitting Eq. (4.19) into real and imaginary parts yields

$$\begin{aligned}\nabla^2\Psi_b^R &\approx \left[A + \frac{1}{a}\left(N_{b-1} - N_b\right)\right]\Psi_b^R, \\ \nabla^2\Psi_b^I &\approx \left[A + \frac{1}{a}\left(N_{b-1} - N_b\right)\right]\Psi_b^I,\end{aligned} \tag{4.21}$$

where

$$A = \frac{\nabla^2\Psi_{b-1}^R\,\Psi_{b-1}^R + \nabla^2\Psi_{b-1}^I\,\Psi_{b-1}^I}{\left(\Psi_{b-1}^R\right)^2 + \left(\Psi_{b-1}^I\right)^2}, \tag{4.22}$$

and N_{b-1} and N_b are as defined in Eq. (4.20).

As discussed in Sec. 4.1, the MSD boundary condition can be expanded out, expressing the $b-1$ Laplacian in terms of the internal scheme in order to be able to evaluate the MSD boundary condition simultaneously with the interior points (when using explicit time-stepping schemes, this is usually unnecessary). As an example, using central-differencing in space for the one-dimensional NLSE, Eq. (4.19) can be expanded as

$$\nabla^2\Psi_b \approx \left[\mathrm{Im}\left(i\,\frac{\Psi_b + \Psi_{b-2}}{\Psi_{b-1}}\right) - 2 + \frac{1}{a}\,\left(N_{b-1} - N_b\right)\right]\Psi_b, \tag{4.23}$$

where N_{b-1} and N_b are as defined in Eq. (4.20).

4.3 NUMERICAL TESTS OF THE MSD BOUNDARY CONDITION

In order to demonstrate the usefulness and advantages of using the MSD boundary condition in the current study, we show some example simulations of the dark coherent structure solutions of the NLSE described in Chapter 2. The MSD boundary condition is compared to a Laplacian-zero (L0) boundary condition defined as $\nabla^2\Psi_b = 0$. The Laplacian-zero boundary condition is chosen for comparison because it is an easy-to-implement boundary condition that has been used for many constant background-density simulations of the NLSE [101].

Every boundary condition including the MSD are only valid given their own assumptions. Therefore, if the modulus-squared of a solution is changing at the boundary, the MSD is not expected to work well, just as if the Laplacian of the wavefunction is far from zero at the boundary, the Laplacian-zero boundary condition is not expected to do well. Therefore, comparisons of which boundary condition is best is very often problem-specific. That being said, comparing the MSD to the L0 is still valuable in that for some problems, both boundary conditions are suitable, allowing for a fair comparison. In addition, since limiting the size of the required grid is very important (especially for higher-dimensional simulations), it is useful to see which boundary condition allows for the use of the smallest (tightest) grid within acceptable accuracy limits.

As in Chapter 3, all our numerical simulations are performed using our GPU-accelerated code package NLSEmagic (we use the non-GPU serial codes in the package for the one-dimensional tests as they run faster due to the small size of the simulations). All simulations are performed using the fourth-order accurate RK4+2SHOC scheme described in Chapter 3.

4.3.1 One-Dimensional Dark Solitons

For the one-dimensional tests, the co-moving dark soliton solution of Eq. (2.5) in Chapter 2 is used. The first test is to compare the error for a steady-state dark soliton

(setting the co-moving velocity c to 0). In such a case, for a large enough domain, the L0 boundary condition can be used since Ψ flattens out as $r \to \infty$. In contrast, the MSD boundary condition should work on any sized domain, since its underlying assumption of constant density is valid anywhere along the steady-state solution. A one-sided second-order differencing (1SD) boundary condition defined in one-dimension as

$$\frac{\partial^2 \Psi}{\partial x^2} \approx \frac{1}{h^2} \left(-\Psi_{b-3} + 4\,\Psi_{b-2} - 5\,\Psi_{b-1} + 2\,\Psi_b \right),$$

is also used for comparison in this case.

We define a radius, r, which represents the distance from the center of the soliton to the edge of the computational domain. Simulations are performed for various lengths of r ranging from $r = 5$ (approximately equal to the width of the soliton) to $r = 25$ (the distance where the boundary value of the soliton is approximately equal to the infinite background density minus machine epsilon $\epsilon \approx 10^{-16}$). The average of the maximum errors in the real and imaginary parts of Ψ over the length of the simulations are recorded. The results are shown in Fig. 4.1. Keeping in mind that $h^4 = 10^{-4}$ and the spatial scheme has an accuracy of $O(h^4)$, it is clear that the MSD boundary condition performs well even when the domain is small. The L0 and 1SD have less error for large r and converge to the error from the exact boundary conditions. This is understandable since as the Laplacian tends towards zero rapidly as r increases, the L0 and 1SD both equate the Laplacian to 0 and have no additional errors associated with them. Even so, the MSD can simulate the solution to acceptable accuracy at a much smaller grid size than the L0 or the 1SD, demonstrating its usefulness in this case.

For the next test, the soliton is given a velocity of $c = 0.5$. Since the L0 assumptions are completely invalid at any domain size, it is not used for comparison (the 1SD is also not used as it fails quickly at any domain size as well). Therefore, the MSD boundary condition is only compared to the exact boundary condition. In this case the domain is set to be a distance r to the left of the initial position of the soliton as before,

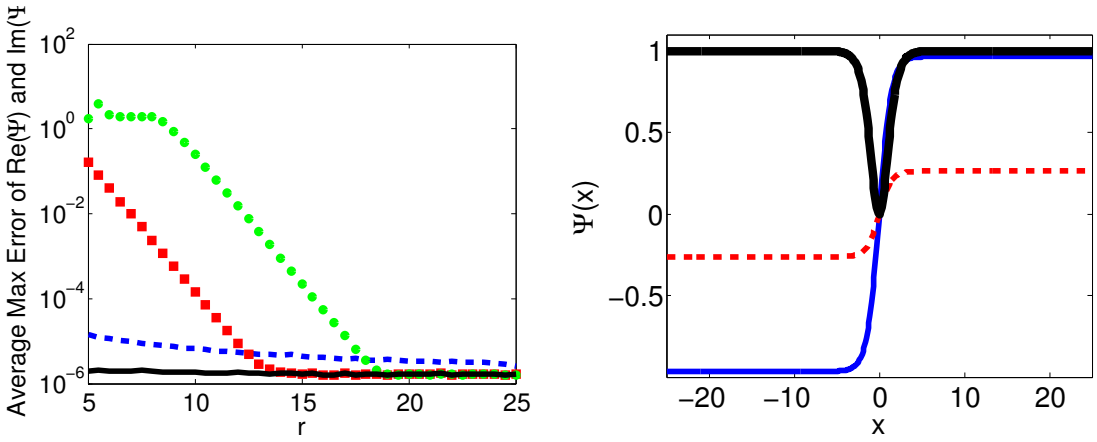


Figure 4.1. Left: Comparisons of errors for simulating the dark-soliton solution (with $c = 0$, $\Omega = -1$, $a = 1$, and $s = -1$) of Eq. (2.5) using MSD (blue dashed line), L0 (red squares), 1SD (green dots), and exact (black solid line) boundary conditions for $r \in [5, 5.5, 6, ..., 25]$. The simulations are run to an end time of $t = 50$ with a time-step of k $= 0.005$ and spatial step of $h = 0.1$. Right: Depiction of the soliton for the maximum domain ($r = 25$) simulation at time $t = 50$. The thin (blue) and dashed (red) lines are the real and imaginary parts of Ψ respectively, while the thick solid (black) line is the modulus-squared, $|\Psi|^2$.

and a distance $r + cT$ to the right (where T is the simulation end-time) in order to account for its movement. In Fig. 4.2, we show the results of the simulations. We see that the MSD boundary condition performs extremely well as long as the soliton is far enough from the boundaries (in this case 'far enough' is about equal to where the background density minus $|\Psi|^2$ at the boundary is approximately h^4). We note that in other tests not presented here using the RK4+CD scheme, it was found that the MSD had *less* error than the exact boundary conditions for large values of r. This observation can possibly be explained by the fact that using exact boundary conditions in fourth-order Runge-Kutta schemes can actually introduce errors in the solution as described in Ref. [67].

4.3.2 Two-Dimensional Dark Vortices

To test the MSD boundary condition in a more complicated setting (and one which is more relevant for the current study), we use the known dynamics of dark vortices in the two-dimensional NLSE described in Sec. 2.3.3 in Chapter 2. It is important to note that the tails of the dark vortices converge to the background density much slower than the

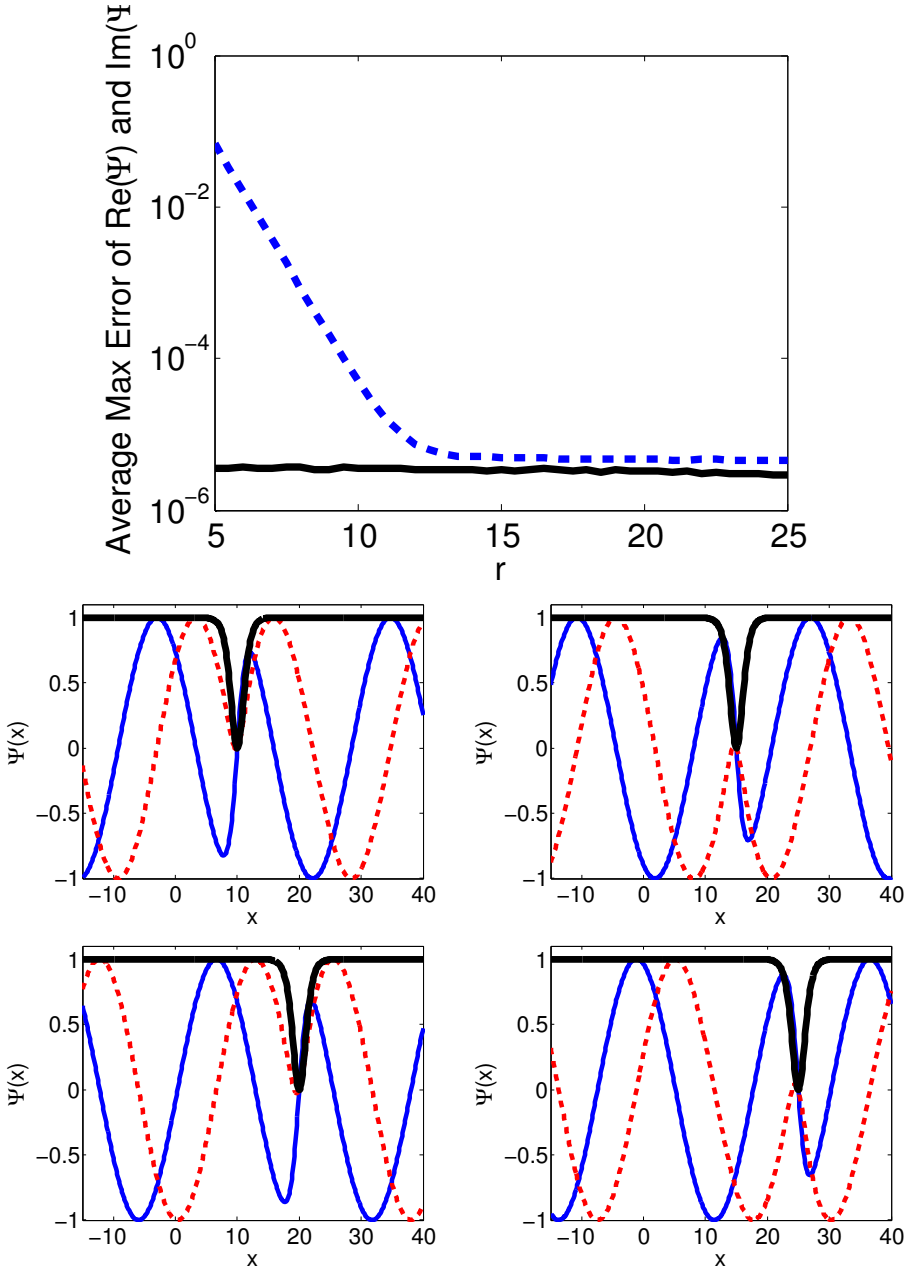


Figure 4.2. Top: Comparisons of errors for simulating the dark-soliton solution of Eq. (2.5) using MSD (blue dashed line), and exact (black solid line) boundary conditions for various domain sizes. The simulation parameters and figure descriptions are the same as in Fig. 4.1 except that here, the velocity is $c = 0.5$. Bottom: Depiction of the soliton during the simulation with $r = 15$ at times $t = 20, 30, 40, 50$ using the MSD boundary condition.

one-dimensional dark solitons of Eq. (2.5). For example, in the one-dimensional dark soliton, we could extend the domain to a size so that the wave-function is at a value of $\sqrt{\Psi_\infty} - \epsilon$ ($\epsilon \approx 10^{-16}$), and this would give a radius of around 25 (for our parameter choices). To get the same boundary value in a two-dimensional vortex of charge $m = 1$, we would require a radius of over *1 million*! Therefore, the ability of the chosen boundary condition to not effect the dynamics of the system on a small grid is vital.

The first test is to simulate a single unitary-charged ($m = 1$) vortex which is known to be a stable steady-state solution to the NLSE [12]. Choosing a moderate domain size (a 120×120 grid with a spatial step-size of $h = 0.25$), we integrate the NLSE for considerable time (up to $T = 50,000$) using both the MSD and L0 boundary conditions. The results are shown in Fig. 4.3. It is easy to see that the L0 boundary condition is only useful for shorter simulations, as it quickly suffers from phase discrepancies which get worse as time progresses, until the point where the solution is distorted badly enough to be unusable. From successive tests it is found that this effect occurs for the L0 boundary condition for even very large domains, but the time of the onset of the distortions is delayed longer as the domain size is increased. In contrast to this, the MSD boundary condition creates *no distortions* in the phase or modulus-squared of the solution for very long simulations (in this case up to $t = 50,000$).

Given a set end-time, it is useful to determine the size of the grid needed to properly simulate the vortex for a given boundary condition. Using the initial vortex solution as representing the 'exact' steady-state solution, we can track the error in the modulus-squared of the solution over the course of the simulations as the grid-size is increased. A radius r is defined as the distance from the center of the vortex to the edge of the domain in the x or y direction (whichever is smallest). Figure 4.4 shows the results of varying r from 5 to 35 for a simulation with an end-time of $t = 300$. It is clearly seen that a much larger grid is required for the L0 boundary condition to be close to the effectiveness of the MSD boundary condition in this case. For example, even at a large

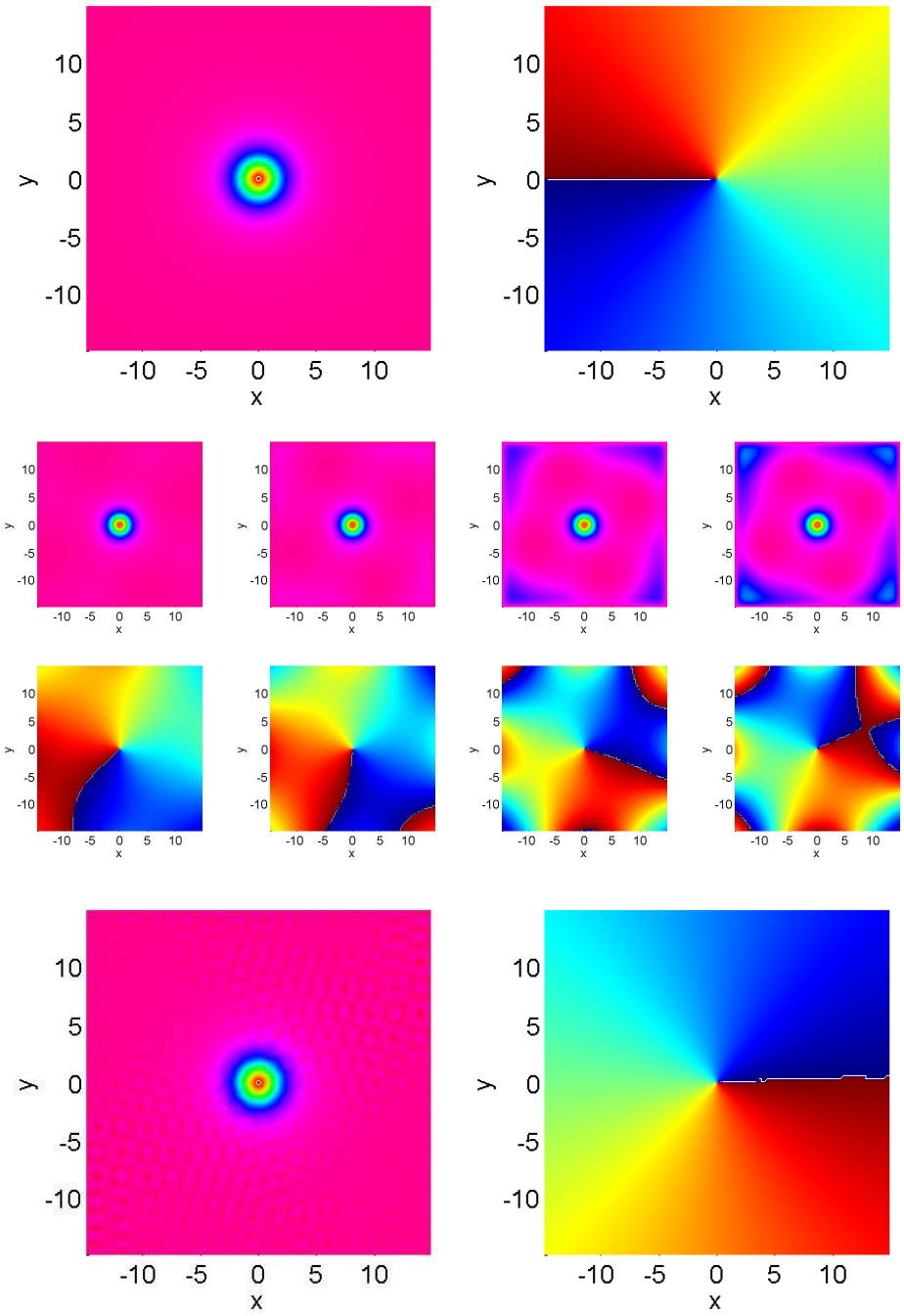


Figure 4.3. Top row: Modulus-squared and phase of the initial condition of a single dark vortex solution. Second (third) row: Snapshots of the density (phase) using the Laplacian-zero boundary condition at times 600**,** 1200**,** 2400**, and** 3000**. Bottom row: Modulus-squared and phase of simulation using the MSD boundary condition at time** $t = 50,000$**. All simulations use a spatial-step of** $h = 0.25$ **and a time-step of** $\mathrm{k} = 0.001$ **on a grid size of** 120×120**.**

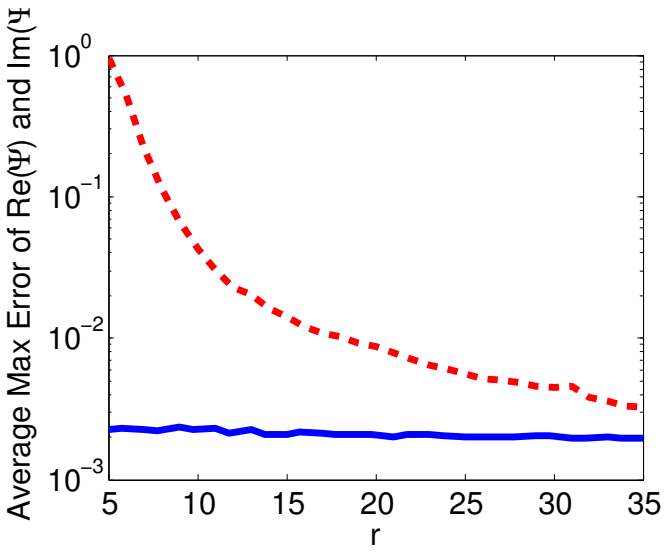


Figure 4.4. Maximum error of the modulus-squared of a steady-state dark vortex using the L0 (red dashed line) and the MSD (solid blue line) boundary conditions for various domain sizes. The initial numerically optimized solution of the vortex is used as the 'true solution' for error comparisons. The simulations were run to an end-time of $t = 300$ with a spatial-step of $h = 0.25$ and time-step k $= 0.01$.

280×280 grid, the L0 boundary condition did not have as low an error as the MSD boundary condition did on a 41×41 grid! Once again, this is understandable considering that the profile of the vortex does not flatten out rapidly as in the one-dimensional case.

As an additional example to test the MSD boundary condition, we simulate two equal-charge vortices whose interaction is known to produce a rotating circular motion of the two vortices orbiting each other [102]. Using a fixed grid size of 171×171, the simulations are run for long times using the L0 and MSD boundary conditions. The positions of the vortices are tracked using the technique described in Chapter 9. The results are shown in Fig. 4.5. We see that once again, using the L0 boundary condition causes a break-down in the dynamics, eventually causing the two vortices to decouple from each other and fling into the boundaries. The MSD boundary condition, by contrast, allows for near-perfect rotational dynamics for indefinite simulation times, even for small grid sizes. It is also noticeable that the period of rotation of the vortices is shorter when using the MSD boundary condition when compared to using the L0 boundary condition.

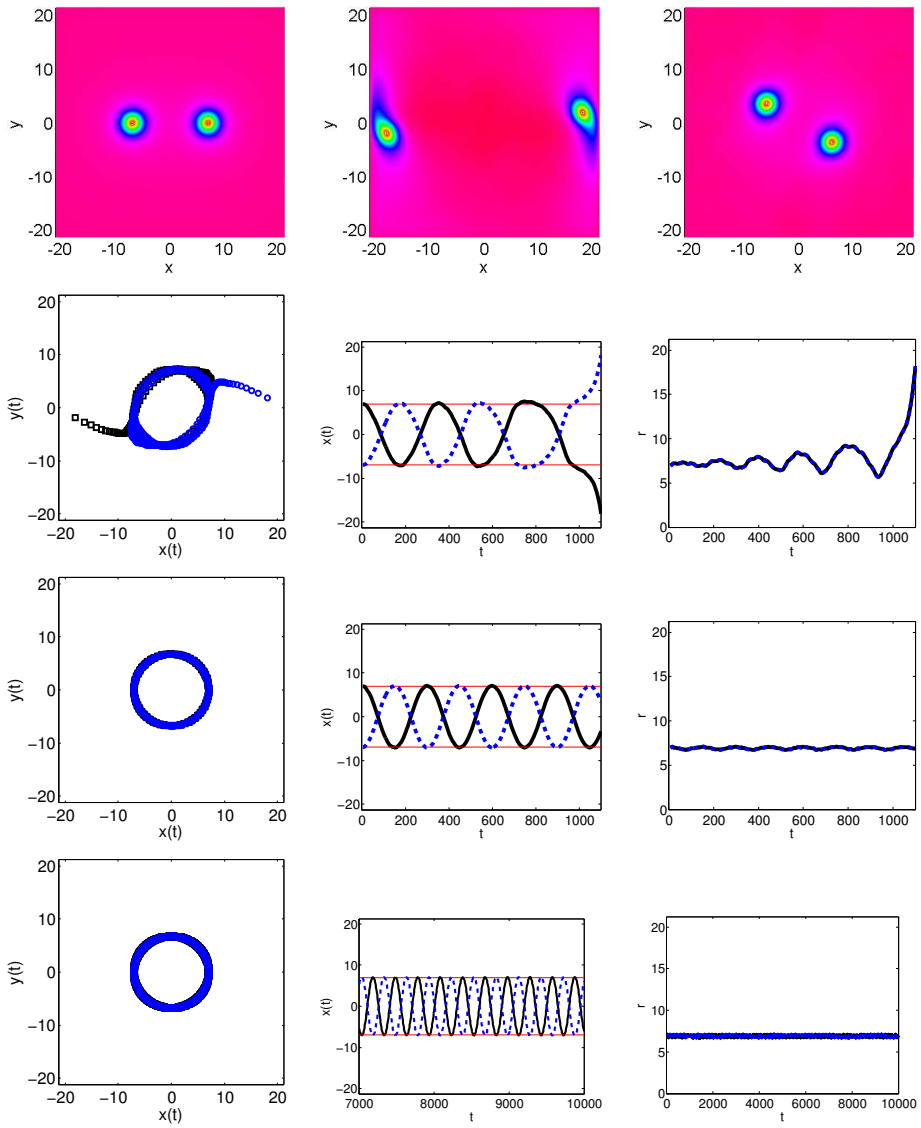


Figure 4.5. Top row, left to right: Modulus-squared of initial condition of two single-charge vortices separated by a distance of 7 from the center of the grid, a snapshot of simulation using the L0 boundary condition at time $t = 1100$, and a snapshot using the MSD boundary condition at time $t = 10,000$. Second row, left to right: Traced positions of the two vortices (square and circles respectively) over the course of a simulation with an end time of $t = 1100$ using L0 boundary conditions, x positions of the two vortices (solid and dashed line respectively) versus time, and the computed radii the two vortices (solid and dashed line respectively) from the center of the grid for an end time of $t = 1100$. Third row: Same description as the second row, but using the MSD boundary condition. Bottom row: Same as in the third row, but with an end-time of $t = 10,000$ (only times $7,000$ through $10,000$ shown in the x versus t plot). In all simulations the spatial-step is $h = 0.25$ and the time-step is $\mathrm{k} = 0.01$ with a grid size of 171×171.

Through further simulations with larger grid sizes, we have observed that the period of rotation converges to the same value (between the two values of the period displayed) for both boundary conditions at approximately the same grid-size (around 250×250). Therefore, the MSD boundary condition performs as well as the L0 in terms of rotational period, but much better in terms of long-term dynamics and minimum grid-size requirements.

If the end-time of the simulation is fixed to be such that the vortices will rotate at least one complete rotation (here we use $t = 480$), we can record the deviation from perfect circular motion as the grid size (given as the distance, d, from the center of the grid to the edge along the x or y direction) is varied. The vortices are tracked during the simulations and the maximum variation of radius compared to a constant radius to the center of the rotating vortices is recorded. The results are shown in Fig. 4.6. We see that the L0 boundary condition requires a very large grid size to capture the correct dynamics (and completely fails for smaller grids), while the MSD is able to capture the dynamics to an acceptable degree on a much smaller grid-size. The discrepancies when using the MSD boundary condition at low grid sizes (up to 20% radius variation) is understandable since at those distances the boundaries become far from steady-state due to the indentations in the background density caused by the motion of the vortices.

4.3.3 Three-Dimensional Dark Vortex Rings

To facilitate the numerical study of vortex rings of Chapters 9 and 10, as well as to further show the usefulness of the MSD boundary condition in multi-dimensional settings, we compare the MSD boundary condition to the L0 within simulations of three-dimensional dark vortex rings in the NLSE.

As discussed in Sec. 2.4 of Chapter 2, dark vortex rings have an intrinsic transverse velocity associated with them [15]. Therefore, in order to run long simulations of the rings (for instance, to study stability under small perturbations, or interactions between multiple co-moving rings), a very large grid size is required. Often, due to the

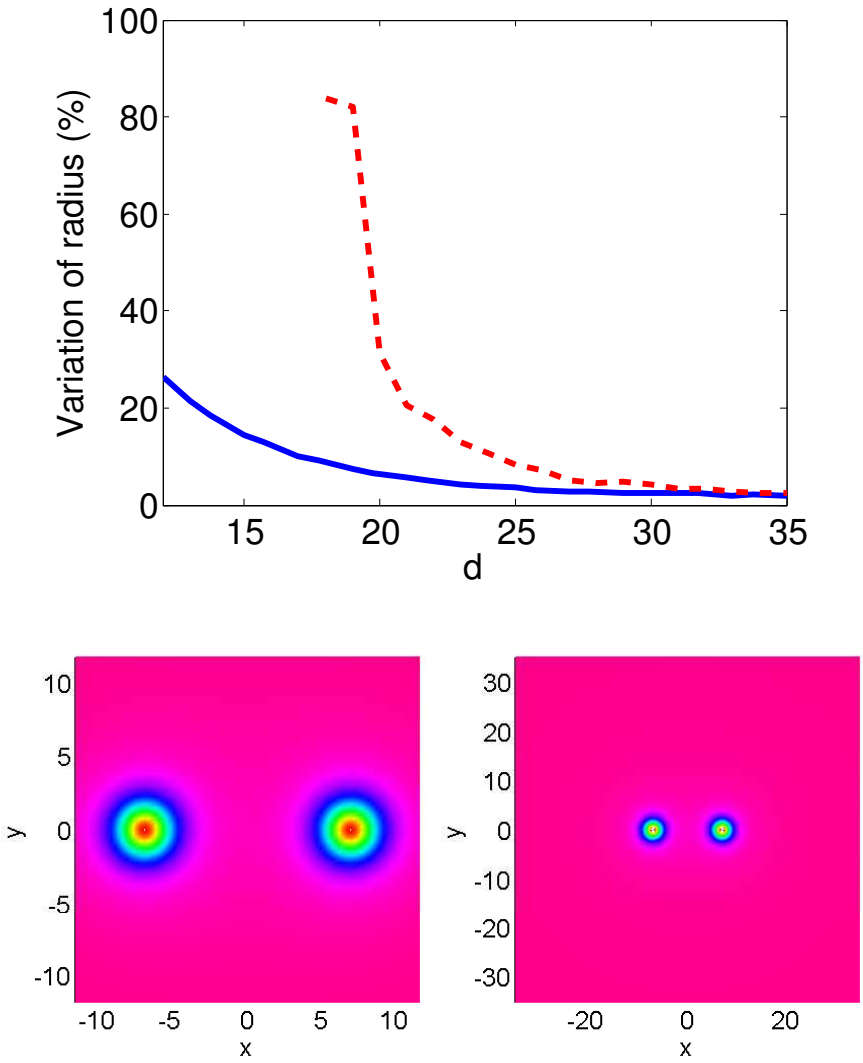


Figure 4.6. Top: Percentage of radius variation when compared to the initial radius of the vortices to the center of the grid as a function of grid size d (defined as the distance from the center of the grid to its edge along the x or y direction) for a simulation with an end-time of $t = 480$ using both the L0 (dashed (red) line) and the MSD (solid (blue) line) boundary conditions. The grid size, d, is varied from 12 (the minimum size required to resolve both vortices adequately) to 35. Results for the L0 boundary condition below $d = 18$ are not shown as the vortices hit the grid wall before the simulation ends. The other simulation parameters are the same as those in Fig. 4.5. Bottom: Modulus-squared of the initial condition for grid-sizes $d = 12$ and $d = 35$.

size of the simulations, a large enough grid is not within the memory limitations of the computers being used for the simulations. To avoid this problem, a vortex ring can be made to be a 'steady-state' by applying a background velocity equal and opposite to the vortex ring's intrinsic velocity (see Chapter 2). By doing this, long-term simulations of the rings can be performed, but with many fewer grid points.

In Fig. 4.7 we show a simulation of a steady-state vortex ring of radius 6 amidst a back-flow using the MSD boundary condition (for details on the generation of the vortex rings, see Chapter 9). As was the case with the co-moving dark soliton of Fig. 4.2, the assumptions underlying the L0 boundary condition are not valid at any grid size, and so it is not tested in this specific case. The vortex ring remains stationary for very long

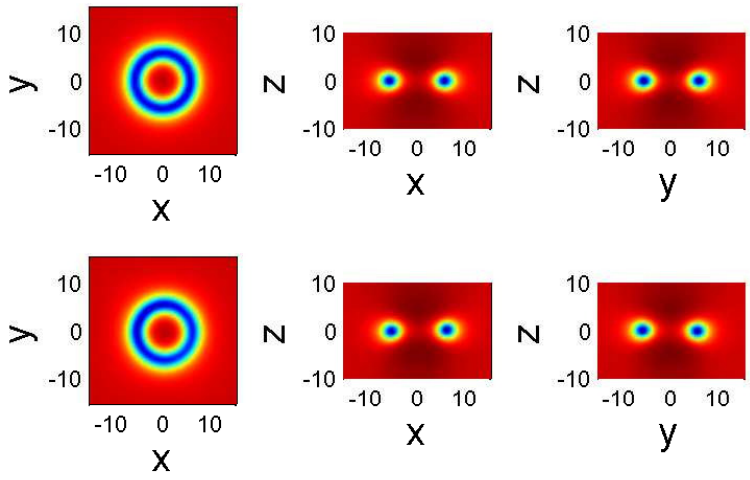


Figure 4.7. Two-dimensional cuts of the modulus-squared of the initial condition of a steady-state vortex ring of radius 6 amidst a back-flow. Bottom row: Two-dimensional cuts of the same vortex ring at simulation time $t = 1500$. The simulation uses a spatial step-size of $h = 0.5$, and a time-step of $k = 0.035$.

simulation times (up to $t = 1500$ in this case), further demonstrating the MSD boundary condition's usefulness in studying co-moving solutions.

As was the case for two-dimensional vortices, when running dynamical simulations of vortex rings, it is important to know how large to set the computational grid in order to minimize the effect of the boundary conditions on the interior dynamics. Since

the vortex rings have a constant transverse velocity (see Sec. 2.4), a good test of the effects of the boundary conditions is to track a vortex ring in a simulation and observe the velocity for various grid sizes. As the size of the grid is expanded, this velocity should converge.

In Fig. 4.8, we show the results of simulating vortex rings of radii 4, 6, and 8 for 50 time units for grid sizes determined by a maximum distance (denoted r_{pad}) from the core of the vortex ring in each direction (taking the travel distance in the z-direction into account). The difference between the velocity of the vortex rings when using the MSD boundary condition is compared to that of using the L0, as well as how each compare to the predicted value given in Sec. 2.4. For details on the generation of the vortex rings and the tracking procedure, see Chapter 9. We see that both the MSD and L0 boundary conditions converge to a constant velocity very close to that of the predicted value given in Sec. 2.4. It is also noteworthy that the convergence happens at roughly the same distance from the vortex ring independent of the vortex ring's radius.

From Fig. 4.8 we notice that the L0 boundary condition converges to a constant velocity more rapidly than the MSD. However, the L0 boundary condition can suffer from additional difficulties with smaller grid sizes, namely the spontaneous creation of a vortex ring due to the L0 boundary condition's problem in maintaining a proper phase structure as was shown in Fig. 4.3. An example of this is shown in Fig 4.9. We see that at such a low r_{pad} value, the MSD boundary condition causes the vortex ring to become more distorted than compared to the L0 boundary condition (however, this should be weighed considering the fact that the vortex ring was simulated for a full 20 time units longer than the case of the L0 boundary condition). More importantly, the L0 boundary condition creates a new, spontaneously generated vortex ring near the top boundary of the grid. Although one would not use such a low r_{pad} value in either case, the error induced in the phase by the L0 boundary condition may greatly effect the dynamics of the vortex rings, especially for long simulations.

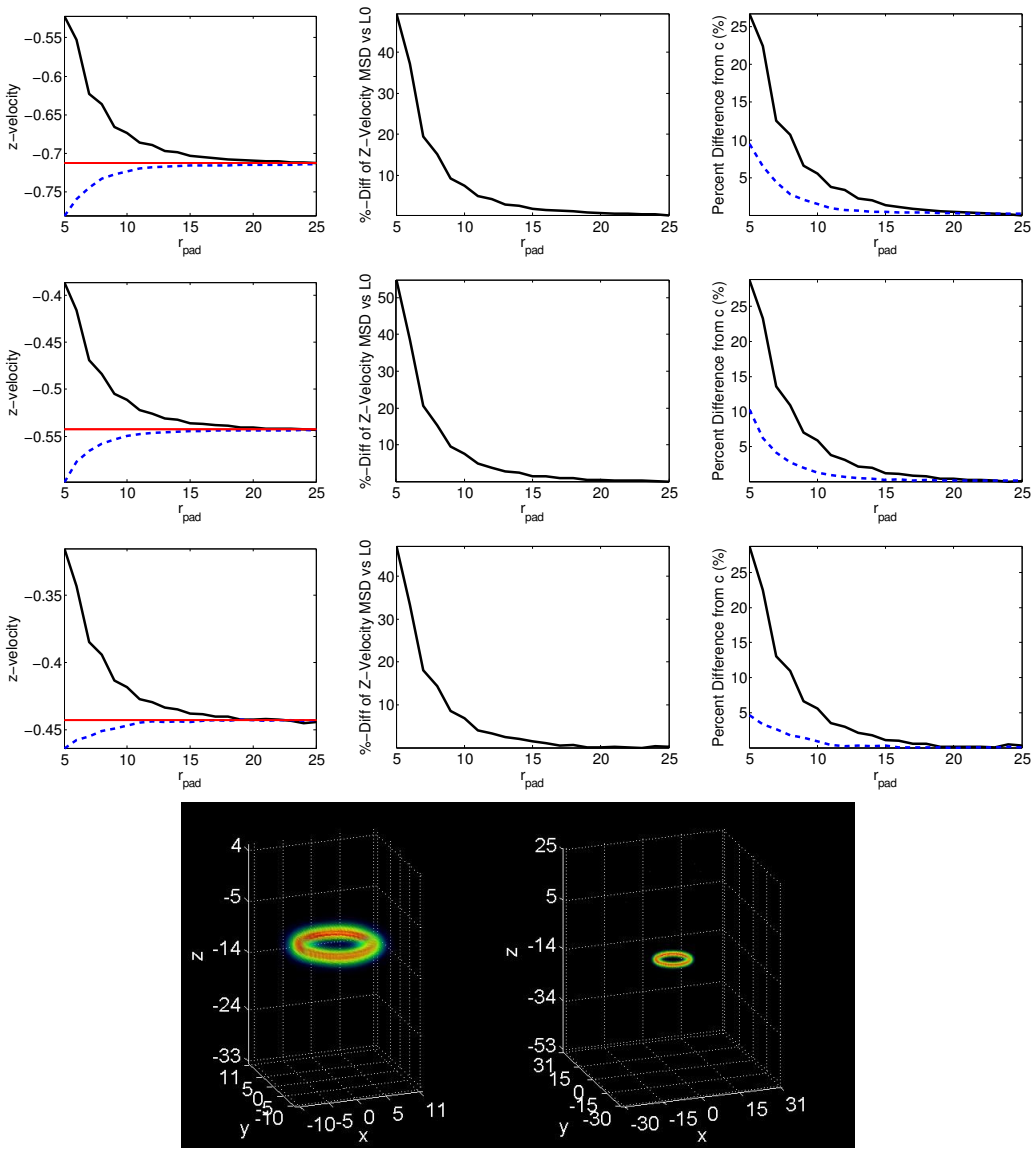


Figure 4.8. Comparison of the velocity of a dark vortex ring in the NLSE over various grid sizes using the MSD and L0 boundary conditions. Left to Right: The observed velocity of the vortex ring versus $r_{\text{pad}} \in [5, 25]$ (the minimum allowed distance distance from the vortex core to the grid boundary allowing for the travel distance) for the MSD (thick solid line), L0 (blue dashed line) boundary conditions (the predicted value of the velocity (thin red line) given in Sec. 2.4 is shown for comparison. The percent-difference between the velocities observed using the MSD and L0 boundary conditions. The percent differences of the velocities observed compared to the velocity predicted in Sec. 2.4. Top-to-bottom: Results for a vortex ring of radius 4, 6, and 8. Bottom row: Volumetric rendering of the vortex ring at $t = 20$ using the MSD boundary condition for $r_{\text{pad}} = 5$ (left) and $r_{\text{pad}} = 25$ (right). The vortex rings are simulated to an end-time of $t = 50$ using a time-step size of $\mathrm{k} = 0.04$ and a spatial-step size of $h = 1/2$.

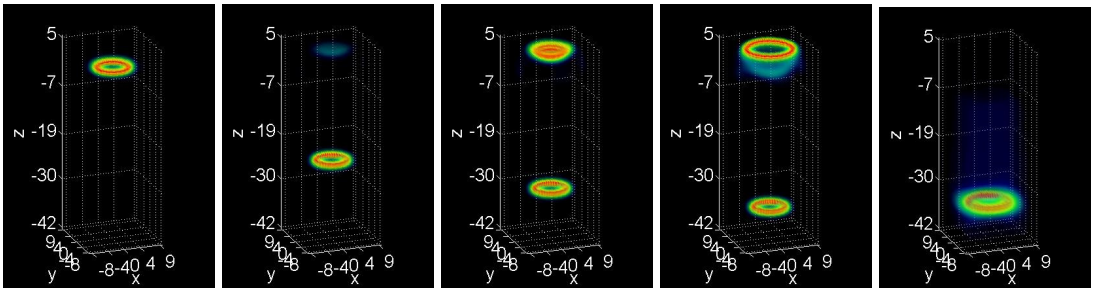


Figure 4.9. Example of spontaneous vortex ring generation when using L0 boundary condition on a small grid. The snapshots of the vortex ring are shown for $r_{\mathbf{pad}} = 5$ at times $t = 0$, $t = 30$, $t = 39$, and $t = 45$. The far right image is the same vortex ring at time $t = 65$ using the MSD boundary condition (a later time was needed to simulate the vortex ring to the same grid position due to the slower velocity of the ring when using the MSD boundary condition as seen in Fig. 4.8). The numerical parameters used are the same as described in Fig. 4.8.

Considering the numerical tests performed in this chapter, we conclude that the MSD boundary condition performs better overall in the simulations of dark vortex rings in the NLSE and therefore choose to use it exclusively in the numerical studies of Chapters 9 and 10.

Now that the details of the boundary condition implementations of our numerical algorithms have been formulated, we now study the numerical stability of the overall finite-difference schemes.

CHAPTER 5

NUMERICAL ALGORITHMS: STABILITY

As discussed in Chapter 3, the only drawback to using explicit finite-difference schemes (such as the RK4) for simulating PDEs is that they are conditionally stable. This means that there is an upper bound on the allowed size of the time-step which is dependent on the spatial-step size. If the time-step is larger than this bound, the scheme is unstable and diverges [66]. Although rough estimates of the stability bound can be found through an inefficient educated guess-and-check, for higher dimensional scenarios, as well as long and/or large simulations, a more refined and predictable stability bound is essential for efficient simulations.

In this chapter, we formulate linearized stability bounds for simulating the NLSE with the RK4+CD and RK4+2SHOC schemes of Chapter 3. The stability bounds are affected by the choice of boundary conditions and we therefore formulate the bounds for all the boundary conditions used in this dissertation which include Dirichlet, modulus-squared Dirichlet (MSD), and Laplacian-zero (L0). We also study the case of periodic boundary conditions as they lead to analytical stability bounds and are another common boundary condition used with the NLSE. Each analysis is done for one, two and three dimensions.

5.1 STABILITY THEORY

Given an initial value problem of a set of linear first-order ODEs (in our case, a method-of-lines explicit PDE finite-difference scheme), one can formulate the matrix notation

$$\frac{\partial \vec{\Psi}}{\partial t} = \mathcal{A}\,\vec{\Psi}, \tag{5.1}$$

where $\mathcal{A}$ contains the coefficients of the right-hand-sides of the ODEs. We now define

$$\vec{p} = \mathbf{k}\,\vec{\lambda}, \tag{5.2}$$

where $\mathbf{k}$ is the time-step size and $\vec{\lambda}$ contains the eigenvalues of $\mathcal{A}$. In our case, the eigenvalues of $\mathcal{A}$ will have the spatial-step size (denoted h) included in them, as well as any parameters of the NLSE. As shown in Ref. [103], for the fourth-order Runge-Kutta scheme, if a vector $\vec{R}(\vec{p})$ is defined whose elements are the polynomials

$$R(p) = 1 + p + \frac{p^2}{2} + \frac{p^3}{6} + \frac{p^4}{24}, \tag{5.3}$$

then the stability of the RK4 scheme is guaranteed if

$$\|\vec{R}(\vec{p})\|_\infty < 1, \tag{5.4}$$

where $\|\ \|_\infty$ denotes the infinity norm defined as $\|\vec{x}\|_\infty = \max\{|x_0|, |x_1|, ..., |x_{N-1}|\}$. Inserting Eq. (5.2) into Eq. (5.3) yields

$$\begin{aligned} |R(\lambda)|^2 = 1 + \frac{1}{576}\,\mathbf{k}^8|\lambda|^8 - \frac{1}{72}\,\mathbf{k}^6|\lambda|^6 + \left(\frac{\mathbf{k}^6|\lambda|^6}{6} - \mathbf{k}^4|\lambda|^4 + 24\right)\,\frac{\mathbf{k}}{12}\,\mathrm{Re}(\lambda) \\ + \left(\mathbf{k}^4|\lambda|^4 + 24\right)\,\frac{\mathbf{k}^2}{12}\,(\mathrm{Re}(\lambda))^2 + \left(\mathbf{k}^2|\lambda|^2 + 4\right)\,\frac{\mathbf{k}^3}{3}\,(\mathrm{Re}(\lambda))^3 + \frac{2\mathbf{k}^4}{3}\,(\mathrm{Re}(\lambda))^4. \end{aligned} \tag{5.5}$$

In Fig. 5.1 we show the stability region for the RK4 scheme given by Eq. (5.4) as well as that for lower-order Runge-Kutta schemes (whose $R(p)$ is defined by progressively truncated versions of Eq. (5.3)) [103]. As we shall show, the eigenvalues of the $\mathcal{A}$ matrix are all purely imaginary (or nearly so) in the case of the nonlinear (and linear) Schrödinger equation. Thus, it can be seen from Fig. 5.1 that the third-order Runge-Kutta is the lowest-order RK scheme that is conditionally stable for the Schrödinger equations (however, as shown in Ref. [94], this is not the case if the real and

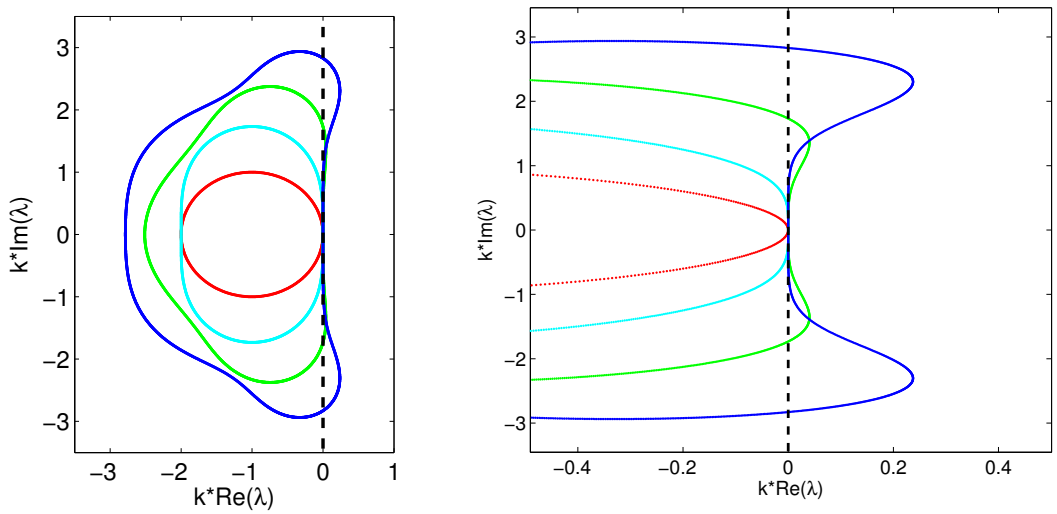


Figure 5.1. Left: Stability regions for Runge-Kutta schemes. Schemes from first-order to fourth-order are shown from center outwards. Right: Magnified view of the same plot near the point where Re$(\lambda) = 0$.

imaginary parts of the NLSE are computed in a staggered time grid, or, as in Ref. [104], an artificial dissipative term is added to the NLSE), with the RK4 yielding a significantly larger bound on $\mathbf{k}$. This is in contrast to similar PDEs such as the heat equation, whose $\mathcal{A}$ matrix eigenvalues are typically all real-valued, in which case even forward differencing (RK1) is conditionally stable.

If, as in our case, Re$(\vec{\lambda}) = \vec{0}$, Eq. (5.5) simplifies greatly and becomes

$$\left|R(\vec{\lambda})\right|^2 = 1 + \frac{1}{576}\,\mathbf{k}^8|\vec{\lambda}|^8 - \frac{1}{72}\,\mathbf{k}^6|\vec{\lambda}|^6, \tag{5.6}$$

in which case, Eq. (5.4) leads to the simple stability bound

$$\mathbf{k} < \frac{\sqrt{8}}{\|\vec{\lambda}\|_\infty}. \tag{5.7}$$

Applying the above stability theory to the NLSE has the obvious problem that the analysis is purely linear, while the NLSE has one (or more) nonlinear terms. A full nonlinear stability analysis is beyond the scope of this dissertation, so instead we linearize

the problem by treating the nonlinearity ($|\Psi|^2$) as a constant value U (the external potential term is usually a constant independent of Ψ at each grid point, and so does not need any special treatment). This has been done previously for the one-dimensional coupled NLSE for fourth-order differencing (in the exclusive case where $s < 0$) in Ref. [93]. Since the value of $|\Psi|^2$ changes over time during the simulation, the linearized stability bound will also change over time. This change in many cases is expected to be small (which we have confirmed in numerical simulations, not reported here) and therefore can be ignored, i.e. one may compute the bound using the initial condition of Ψ (and $V(\mathbf{r})$) and just leave a few percent leeway to cover any changes. This is especially true in the repulsive case ($s < 0$) where most situations have a constant-density background (or maximum background) and the dynamics do not cause the maximum background value to change significantly (for example, when simulating coherent structures such as vortex rings, most of the dynamics are translations of the initial condition with little change in overall global structure). In attractive cases ($s > 0$), blow-up can occur which can alter the stability bound greatly, causing the simulation to crash (although in such a case the wavefunction is exploding towards infinity, which most finite-difference schemes cannot handle anyways). Many times, simulations of a steady-state or near-steady-state in the modulus-squared with a constant potential are performed (such as the steady-state solitons and vortices shown in Chapter 2). In such situations, the linearized stability bounds will be (nearly) exact.

It is also useful to formulate stability bounds for the basic linear Schrödinger equation (LSE) (where $s = 0$ and $V(\mathbf{r}) = 0$). In addition to providing bounds for the LSE, as will be discussed below, the results can also be used as practical estimates of the stability bounds for the NLSE (the discrepancy can often be solved by lowering the bound by a few percent).

In order to simplify the analysis, we first rewrite Eq. (5.1) as

$$\frac{\partial \vec{\Psi}}{\partial t} = \mathcal{A}\vec{\Psi} = \frac{i\,a}{h^2}\,A\vec{\Psi},$$

where h is the step-size of the spatial finite-difference scheme being used. Then, assuming all eigenvalues of A are real-valued, the stability condition of Eq. (5.7) becomes

$$\mathbf{k} < \frac{\sqrt{8}}{\|\vec{\lambda}_A\|_\infty} \frac{h^2}{a}. \tag{5.8}$$

In order to be able to use the stability bound of Eq. (5.8), we must first confirm that all eigenvalues of A are purely real (or nearly so) for each scheme/boundary condition combination. In cases where the eigenvalues are not able to be easily computed analytically, we show that the A matrix's eigenvalues are a set of boundary values with the remaining eigenvalues being those of a symmetric matrix denoted A'. Then by Thm. 1, it is known that all the eigenvalues of A are real.

Theorem 1 (Ref. [105])**.** *The eigenvalues of a real symmetric matrix are real.*

Once it has been established that Eq. (5.8) can be used, in order to get an upper-bound on $\mathbf{k}$, we require an upper-bound on the maximum absolute eigenvalue of A. Due to the sparsity and diagonal dominance of A, a good estimate of the upper-bound can be found using the Gershgorin circle theorem (Def. 1 and Thm. 2).

Definition 1 (Ref. [106])**.** *Let A be a square complex matrix. Around every element a_{ii} on the diagonal of the matrix, a circle with radius equal to the sum of the norms of the other elements in the same row ($\sum_{j\neq i} |a_{ij}|$) is known as a* Gershgorin disc.

Theorem 2 (Ref. [106])**.** *Every eigenvalue of a square complex matrix A lies in one of its Gershgorin discs.*

Since every eigenvalue must be contained in a Gershgorin disk, by finding the maximum absolute value of the limits of the disks will yield an upper-bound on the maximum modulus of the eigenvalues of A.

In the one-dimensional LSE case with no external potential and periodic boundary conditions, the A matrix becomes circulant as defined by Def. 2. In this case, the

eigenvalues can be computed analytically by Thm. 3. The upper-bound is then taken by finding the limit of the maximum eigenvalue as the size of the matrix goes to infinity.

Definition 2 (Ref. [107]). *A circulant matrix is a square $N \times N$ matrix C that can be fully specified by one vector, $\vec{c} = \{c_0, c_1, ..., c_{N-1}\}$, which appears as the first column of C. The remaining columns of C are each cyclic permutations of the vector with the offset equal to the column index.*

Theorem 3 (Ref. [107]). *The eigenvalues of a circulant matrix are given by*

$$\lambda_j = c_0 + c_{N-1}\,\omega_j + c_{N-2}\,\omega_j^2 + ... + c_1\,\omega_j^{N-1}, \qquad j = 0, ..., N-1,$$

where

$$\omega_j = \exp\left(\frac{2\pi\, i\, j}{N}\right).$$

5.2 Boundary Conditions

Since boundary conditions of the spatial differencing in a PDE like the NLSE have the potential to alter the stability of a scheme dramatically, it is necessary to have stability results for each specific boundary condition one would like to use. We use the same notation to describe the boundary points as in Chapter 4, (i.e. we use the subscript b to represent any boundary point, and $b-1$ to represent the grid position one point inward from the boundary in the normal direction).

For use with the stability analysis, it is desirable to formulate each boundary condition in terms of the temporal derivative in the form

$$\left.\frac{\partial \Psi}{\partial t}\right|_b = \frac{i\,a}{h^2} B_b \Psi_b, \tag{5.9}$$

and in terms of the spatial Laplacian in the form

$$\nabla^2 \Psi_b = \frac{1}{h^2} D_b \Psi_b, \tag{5.10}$$

where B_b and D_b are assumed to be real-values constants (possibly differing per boundary point) and defined based on the specific boundary condition being used. For periodic boundary conditions (or linear one-sided conditions not discussed here), these forms are not applicable. Writing the boundary conditions in the forms of Eq. (5.9) and Eq. (5.10) allows them to be expressed in the A matrix as a single real-valued entry (B_b), and in the case of the fourth-order 2SHOC differencing, the near-boundary interior points will simply contain D_b in their formulation.

As discussed in Chapter 4, Dirichlet boundary conditions are defined as

$$\Psi_b = C,$$

where C is a constant. In terms of the temporal derivative of the NLSE, this condition is

$$\left.\frac{\partial \Psi}{\partial t}\right|_b = 0,$$

in which case $B_b = 0$ in Eq. (5.9). When inserted into the NLSE, this condition in terms of the Laplacian is given by

$$\nabla^2 \Psi_b = -\frac{1}{a}(s|\Psi_b|^2 - V_b)\Psi_b,$$

and therefore $D_b = -h^2/a(s|\Psi_b|^2 - V_b)$ in Eq. (5.10).

The modulus-squared Dirichlet (MSD) boundary condition formulated in Chapter 4 is defined by having the modulus-squared of the wavefunction to be constant at the boundaries,

$$|\Psi_b|^2 = C,$$

where C is a constant. The MSD boundary condition is given in terms of the temporal derivative of the NLSE as

$$\Psi_{t,b} \approx i \,\mathrm{Im}\left[\frac{\Psi_{t,b-1}}{\Psi_{b-1}}\right] \Psi_b. \tag{5.11}$$

where $\partial\Psi_{b-1}/\partial t$ is computed by the interior scheme first, and then used to compute the boundary values. Using the MSD boundary condition gives $B_b = (h^2/a)\text{Im}\left[\frac{\partial\Psi}{\partial t}\big|_{b-1}\frac{1}{\Psi_{b-1}}\right]$, which is nonlinear, and not a constant independent of Ψ. As shown in Chapter 4, due to the underlying assumptions of the MSD boundary condition, Eq. (5.11) can be viewed as

$$\left.\frac{\partial\Psi}{\partial t}\right|_b \approx i\,\Omega_{b-1}\Psi_b,$$

where Ω_{b-1} is the real-valued frequency of the solution near the boundary. Thus, B_b would have the form $B_b = (h^2/a)\Omega_{b-1}$. Therefore, we can linearize the MSD boundary condition by treating the B_b term as a constant (which can change over the course of the simulation, similar to the nonlinearity of the NLSE).

When inserted into the NLSE, the MSD boundary condition of Eq. (5.11) yields

$$\nabla^2\Psi_b \approx \left[\text{Im}\left(i\,\frac{\nabla^2\Psi_{b-1}}{\Psi_{b-1}}\right) + \frac{1}{a}\,(N_{b-1} - N_b)\right]\Psi_b, \tag{5.12}$$

where

$$N_b = s\,|\Psi_b|^2 - V_b, \qquad N_{b-1} = s\,|\Psi_{b-1}|^2 - V_{b-1}. \tag{5.13}$$

and therefore $D_b = h^2\left[\text{Im}\left(i\,\frac{\nabla^2\Psi_{b-1}}{\Psi_{b-1}}\right) + \frac{1}{a}\,(N_{b-1} - N_b)\right]$. This too is a nonlinear, non-constant term, and so must be treated as a constant in the same manner as the nonlinearity of the NLSE.

The Laplacian-zero boundary condition is defined by

$$\nabla^2\Psi_b = 0,$$

and therefore $D_b = 0$. In terms of the time-derivative of the NLSE, the L0 boundary condition is given as

$$\left.\frac{\partial \Psi}{\partial t}\right|_b = i\left(s|\Psi_b|^2 - V_b\right)\Psi_b,$$

making $B_b = (h^2/a)(s|\Psi_b|^2 - V_b)$. This condition is as easy to implement as the Dirichlet, and can be useful in many situations.

Table 5.1. Boundary condition terms for use with stability analysis.

Boundary Condition	B_b	D_b
Dirichlet	0	$\frac{h^2}{a}(V_b - s\|\Psi_b\|^2)$
Laplacian-zero	$\frac{h^2}{a}(s\|\Psi_b\|^2 - V_b)$	0
MSD	$\frac{h^2}{a}\text{Im}\left[\frac{\Psi_{t,b-1}}{\Psi_{b-1}}\right]$	$h^2\left[\text{Im}\left(i\,\frac{\nabla^2\Psi_{b-1}}{\Psi_{b-1}}\right) + \frac{1}{a}\,(N_{b-1} - N_b)\right]$

To assist the stability analysis, a summary of the values of B_b and D_b for all the mentioned boundary conditions are given in Table. 5.1 for future reference. Many other boundary conditions exist for simulating the NLSE, in which case the analysis shown in this chapter can be adapted to the other boundary conditions.

5.3 ONE-DIMENSIONAL STABILITY ANALYSIS

In the one-dimensional cases we analyze all four boundary conditions mentioned in Sec. 5.2. As stated, periodic boundary conditions yield a matrix where (in the linear case with $s = 0$ and $\vec{V}(\mathbf{r}) = 0$) the eigenvalues can be computed analytically. This allows the results obtained using the upper-bound methods (which we use with other boundary conditions) to be compared with the true eigenvalues giving an idea of how accurate they are.

5.3.1 Second-Order Central Difference

The second-order central difference is one dimension is given by

$$\nabla^2\Psi_i = \left.\frac{\partial^2\Psi}{\partial x^2}\right|_i \approx \frac{\Psi_{i+1} - 2\Psi_i + \Psi_{i-1}}{h^2},$$

and when implemented into the A matrix, forms a matrix which is tridiagonal (except for the two boundary condition rows).

5.3.1.1 Periodic Boundary Conditions

In order to obtain analytic expressions for the eigenvalues of A, we start with the LSE case with no external potential and periodic boundary conditions. This yields the matrix

$$A = \begin{bmatrix} -2 & 1 & 0 & 0 & 1 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 1 & 0 & 0 & 1 & -2 \end{bmatrix},$$

which, as per Def. 2, is a circulant matrix with $\vec{c} = \{-2, 1, 0, ..., 0, 1\}$. Also, since A is a real-valued symmetric matrix, by Thm. 1, all eigenvalues are real and therefore the stability criteria of Eq. (5.8) can be used. By Thm. 3, the eigenvalues of A are given by

$$\lambda_j = -2 + \exp\left[\frac{2\pi ij}{N}\right] + \exp\left[\frac{2\pi ij(N-1)}{N}\right], \qquad j \in \{0, ...N-1\}.$$

The maximum value of $|\lambda_j|$ occurs either at $j = N/2$ if N is even, or $j = (N \pm 1)/2$ if N is odd. For N even-valued we have

$$|\lambda|_{\max} = \left|-2 + \exp\left[\pi i\right] + \exp\left[\pi i\right]^{N-1}\right|,$$

which yields

$$|\lambda|_{\max} = 4.$$

For N odd-valued we have

$$|\lambda|_{\max} = \left|-2 - (-1)^{1/N} + (-1)^N (-1)^{-1/N}\right|,$$

which yields

$$|\lambda|_{\max} = \left|-2 - 2\,\cos\left(\frac{\pi}{N}\right)\right|.$$

Taking $N \to \infty$, the maximum bound on the maximum absolute eigenvalue becomes

$$|\lambda|_{\max} < 4.$$

We therefore have an upper bound on the maximum absolute eigenvalue which, for even-valued N, is guaranteed to *be* one of the eigenvalues. The stability criteria of Eq. (5.8) is then formulated as

$$\mathbf{k} < \frac{\sqrt{8}}{4}\frac{h^2}{a}. \tag{5.14}$$

In the general case where $s \neq 0$ and/or $V(\mathbf{r}) \neq 0$, the A matrix is no longer circulant (since the values of the nonlinearity or external potential vary over the diagonal of A). To get a bound on the maximum absolute eigenvalue, we make use of Thm. 2. The matrix A has N Gershgorin disks, since each diagonal entry of A can be unique, but each disk has the same radius ($r = 2$). Also, since the diagonal entries can in theory take on any value, all Gershgorin disk limits must be examined. This yields the stability bound

$$\mathbf{k} < \frac{\sqrt{8}}{\max\{\|\vec{L}\|_\infty, \|\vec{L} - 4\|_\infty\}}\,\frac{h^2}{a}, \tag{5.15}$$

where we have defined the elements of $\vec{L}$ to be

$$L_i = \frac{h^2}{a}(s\,|\Psi_i|^2 - V_i), \tag{5.16}$$

where the index i spans over the entire grid. It is important to note that all values of $\vec{L}$ are $O(h^2)$. Thus, for $h \ll 1$, and reasonable values of $|\Psi|^2$ and $\vec{V}$, the linear bound of Eq. (5.14) should be very close to the true bound of the nonlinear problem.

If we set $\vec{L} = 0$ in Eq. (5.15), we recover the bound in Eq. (5.14). This shows that (in this case at least), using the Gershgorin circle theorem yields the true bound on the eigenvalues of A.

5.3.1.2 DIRICHLET, MSD, AND L0 BOUNDARY CONDITIONS

As shown in Sec. 5.2, Dirichlet, Laplacian-zero, and modulus-squared Dirichlet boundary conditions can all be viewed as single entries in the boundary value rows of the A matrix, denoted as B_b. As shown there, the values of B_b are real-valued and their values for each boundary condition were given in Table 5.1. Using such a formulation, the A matrix becomes

$$A = \begin{bmatrix} B_0 & 0 & 0 & 0 & 0 \\ 1 & L_1 - 2 & 1 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 1 & L_{N-2} - 2 & 1 \\ 0 & 0 & 0 & 0 & B_{N-1} \end{bmatrix}.$$

In order to use the simple stability criteria of Eq. (5.8), we once again need to show that all eigenvalues of A are purely real. The A matrix is no longer symmetric, however it is easy to see that B_0 and B_{N-1} are eigenvalues of A, and the remaining eigenvalues of A

are equivalent to the eigenvalues of the matrix A' defined as

$$A' = \begin{bmatrix} L_1 - 2 & 1 & 0 & 0 & 0 \\ 1 & L_2 - 2 & 1 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 1 & L_{N-3} - 2 & 1 \\ 0 & 0 & 0 & 1 & L_{N-2} - 2 \end{bmatrix}.$$

Since A' is real-valued symmetric, we can use the stability bound of Eq. (5.8).

We now need to find an upper bound on the absolute value of the eigenvalues of A'. We use the Gershgorin circle theorem to find all unique Gershgorin disks and take the limits of the disks to find the bounds on the absolute eigenvalues. Many of the Gershgorin disks are similar, differing only in the value of L_i of the specific row. Therefore, each disk of different centers and radii has a subset of L_i values relevant to it. Although in the current one-dimensional setting it is simple to define the subsets, in higher-dimensional settings, it can become burdensome to separate out each subset of $\vec{L}$ relevant to each Gershgorin disk of the same center and radius. Therefore, for practicality purposes, we define our bounds using all possible values of L_i for each Gershgorin disk center and radius. This may make the resulting stability bound slightly higher than necessary in certain cases, but this is outweighed by the ease-of-use of the simplified bounds. The unique forms of the Gershgorin disks of A' are shown in Table 5.2. The resulting general

Table 5.2. Unique forms of the Gershgorin disk centers (a_{ii}) and radii (r_i) for the A' matrix of the one-dimensional second-order central difference scheme.

a_{ii}	$r_i = \sum_{i \neq j} \lvert a_{ij} \rvert$
$L_i - 2$	1
$L_i - 2$	2

stability bounds are

$$\mathbf{k} < \frac{\sqrt{8}}{\max\{\|\vec{B}\|_\infty, \|\forall L_i, L_i - \vec{G}\|_\infty\}} \frac{h^2}{a}, \tag{5.17}$$

where $\vec{B}$ are all boundary condition values (in this case B_0 and B_{N-1}), and $\vec{G}$ is defined as

$$\vec{G} = \{4, 3, 1, 0\}\,. \tag{5.18}$$

In general, all possible values of $\vec{G}$ must be taken into consideration since there is no theoretical restriction on what values $\vec{L}$ can take. However, in certain specific circumstances, some of the values of $\vec{G}$ can be ignored (for example, when $s \leq 0$ and $V(\mathbf{r}) \geq 0$, only the largest magnitude value in $\vec{G}$ is needed).

5.3.2 Fourth Order Central Difference

The standard fourth order central difference scheme is given by

$$\nabla^2 \Psi_i = \left.\frac{\partial^2 \Psi}{\partial x^2}\right|_i \approx \frac{-\Psi_{i+2} + 16\Psi_{i+1} - 30\Psi_i + 16\Psi_{i-1} - \Psi_{i-2}}{12\,h^2}. \tag{5.19}$$

The stability analysis follows directly from the second-order case. The only major difference is that since the fourth order stencil is five points wide, the grid points near the boundary may need special consideration for the different boundary conditions. For our purposes here, we use the two-step high-order compact (2SHOC) version of the fourth-order scheme as described in Chapter 3, in which case the near-boundary points can be formulated by combining the two steps of the 2SHOC scheme.

5.3.2.1 PERIODIC BOUNDARY CONDITION

In the periodic case, no special attention is needed near the boundaries, and the A matrix in the LSE case with no external potential is

$$A = \begin{bmatrix} -15/6 & 4/3 & -1/12 & 0 & 0 & -1/12 & 4/3 \\ 4/3 & -15/6 & 4/3 & -1/12 & 0 & 0 & -1/12 \\ -1/12 & 4/3 & -15/6 & 4/3 & -1/12 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & -1/12 & 4/3 & -15/6 & 4/3 & -1/12 \\ -1/12 & 0 & 0 & -1/12 & 4/3 & -15/6 & 4/3 \\ 4/3 & -1/12 & 0 & 0 & -1/12 & 4/3 & -15/6 \end{bmatrix},$$

which is a circulant matrix, and its eigenvalues are therefore

$$\begin{aligned} \lambda_j = &-\frac{15}{6} + \frac{4}{3}\exp\left[\frac{2\pi ij}{N}\right] - \frac{1}{12}\exp\left[\frac{4\pi ij}{N}\right] \\ &- \frac{1}{12}\exp\left[\frac{2(N-2)\pi ij}{N}\right] + \frac{4}{3}\exp\left[\frac{2(N-1)\pi ij}{N}\right]. \end{aligned}$$

The maximum absolute value once again occurs occurs at either $j = N/2$ if N is even, or $j = (N \pm 1)/2$ if N is odd. For N even-valued we have

$$\lambda_{N/2} = -\frac{15}{6} - \frac{4}{3} - \frac{1}{12} - \frac{1}{12}(-1)^{N-2} + \frac{4}{3}(-1)^{N-1} = -\frac{16}{3}.$$

For N odd, we have

$$\lambda_{(N+1)/s} = -\frac{15}{6} - \frac{4}{3}\left((-1)^{1/N} + (-1)^{-1/N}\right) - \frac{1}{12}\left((-1)^{2/N} + (-1)^{-2/N}\right),$$

which yields

$$\lambda_{(N+1)/s} = -\frac{15}{6} - \frac{4}{3}\left(2\cos\left(\frac{\pi}{N}\right)\right) - \frac{1}{12}\left(2\cos\left(\frac{2\pi}{N}\right)\right).$$

As $N \to \infty$, $|\lambda| \to \frac{16}{3}$, which is the same bound as the N-even case. Thus, the stability bound is given by

$$\mathbf{k} < \left(\frac{3}{4}\right)\frac{\sqrt{8}}{4}\frac{h^2}{a}, \tag{5.20}$$

which we note is only a 25% reduction of the second-order bound of Eq. (5.14). In the general case where $\vec{L} \neq 0$, the bound becomes

$$\mathbf{k} < \frac{6\sqrt{2}}{\max\{\|3\vec{L} - 16\|_\infty, \|3\vec{L} + 1\|_\infty\}}\frac{h^2}{a}. \tag{5.21}$$

If $s \leq 0$ and $V(\mathbf{r}) \geq 0$, the first term in the denominator is the maximum of the two terms, and the resulting stability bound is equivalent to that found in Ref. [93] for using the RK4 scheme and fourth-order spatial differencing with the coupled NLSE.

5.3.2.2 DIRICHLET, MSD, AND L0 BOUNDARY CONDITIONS

As per Sec. 5.2, we formulate all three boundary conditions in terms of a B_b entry in the A matrix. As discussed, an important issue is that we need to handle the grid points near the boundary due to the width of the scheme. A common way of dealing with the closest-interior points is to compute the Laplacian at those points using second-order differencing, however this can lead to the overall scheme becoming second-order. However, since we are using the 2SHOC version of the fourth-order differencing, we can derive the closest-interior points which, if the assumptions of the chosen boundary conditions hold, should maintain fourth-order accuracy. In one dimension, the 2SHOC

scheme is defined as Eqs. (3.9) and (3.10) in Chapter 4, reproduced here:

$$1) \qquad D_i = \frac{1}{h^2}\left(\Psi_{i+1} - 2\Psi_i + \Psi_{i-1}\right), \tag{5.22}$$

$$2) \qquad \nabla^2\Psi_i \approx \frac{7}{6}D_i - \frac{1}{12}\left(D_{i+1} + D_{i-1}\right). \tag{5.23}$$

In the first step, the second-order Laplacian is computed with the chosen boundary condition applied to it. Next, the result is used to compute the fourth-order Laplacian. As mentioned in Chapter 3, this two-step scheme is equivalent to the standard wide-stencil of Eq. (5.19) for the interior points. We use the form of Eq. (5.10) for the boundary conditions on the Laplacian, and after combining the steps of Eq. (5.22) and Eq. (5.23), we get the matrix

$$A = \begin{bmatrix}
B_0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{14-D_0}{12} & L_1 - \frac{29}{12} & \frac{4}{3} & -\frac{1}{12} & 0 & 0 & 0 \\
-\frac{1}{12} & \frac{4}{3} & L_2 - \frac{15}{6} & \frac{4}{3} & -\frac{1}{12} & 0 & 0 \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
0 & 0 & -\frac{1}{12} & \frac{4}{3} & L_{N-3} - \frac{15}{6} & \frac{4}{3} & -\frac{1}{12} \\
0 & 0 & 0 & -\frac{1}{12} & \frac{4}{3} & L_{N-2} - \frac{29}{12} & \frac{14-D_{N-1}}{12} \\
0 & 0 & 0 & 0 & 0 & 0 & B_{N-1}
\end{bmatrix}$$

The B_b and D_b terms for each boundary condition are once again given in Table 5.1. As in Sec. 5.3.1.2, the A matrix is not symmetric and has eigenvalues equal to $\vec{B}$. The remaining

eigenvalues are those of the matrix A' defined as

$$A' = \begin{bmatrix} L_1 - \frac{29}{12} & \frac{4}{3} & -\frac{1}{12} & 0 & 0 & 0 & 0 \\ \frac{4}{3} & L_2 - \frac{15}{6} & \frac{4}{3} & -\frac{1}{12} & 0 & 0 & 0 \\ -\frac{1}{12} & \frac{4}{3} & L_3 - \frac{15}{6} & \frac{4}{3} & -\frac{1}{12} & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & -\frac{1}{12} & \frac{4}{3} & L_{N-4} - \frac{15}{6} & \frac{4}{3} & -\frac{1}{12} \\ 0 & 0 & 0 & -\frac{1}{12} & \frac{4}{3} & L_{N-3} - \frac{15}{6} & \frac{4}{3} \\ 0 & 0 & 0 & 0 & -\frac{1}{12} & \frac{4}{3} & L_{N-2} - \frac{29}{12} \end{bmatrix},$$

which is once again real-symmetric so the bounds of Eq. (5.8) can be used. It is interesting to note that the values of D_b do not appear in any of the eigenvalues of A'.

Table 5.3. Unique forms of the Gershgorin disk centers (a_{ii}) and radii (r_i) for the A' matrix of the one-dimensional fourth-order 2SHOC scheme.

a_{ii}	$r_i = \sum_{i \neq j} \lvert a_{ij} \rvert$
$L_i - 5/2$	$11/4$
$L_i - 5/2$	$17/6$
$L_i - 29/12$	$17/12$

The unique forms (see the discussion in Sec. 5.3.1.2) of the Gershgorin disk centers and radii of A' are shown in Table 5.3. The full stability bound is the same form as Eq. (5.17), but with $\vec{G}$ defined as

$$\vec{G} = \frac{1}{12} \times \{64, 63, 46, 12, -3, -4\}. \tag{5.24}$$

Once again, in the most general case, all values of Eq. (5.24) must be considered in finding the maximum allowed time-step value.

5.4 TWO-DIMENSIONAL STABILITY ANALYSIS

In higher dimensions, the A matrix is formed by unwrapping the solution into a one-dimensional vector and then formulating the scheme matrix accordingly.

In Sections 5.3.1.1 and 5.3.2.1 we noted that the stability bounds given using the Gershgorin circle theorem were equivalent to those obtained analytically for the linear case with periodic boundary conditions. We therefore justify relying exclusively on the Gershgorin theorem for higher dimensions, and focus on the stability bounds for Dirichlet, MSD, and Laplacian-zero boundary conditions (since the periodic boundary condition bounds will be a subset of the bounds computed for the other boundary conditions).

5.4.1 Second-Order Central Difference

The second-order central difference scheme in two dimensions is given by

$$\nabla^2\Psi_{i,j} = \left.\frac{\partial^2\Psi}{\partial x^2}\right|_{i,j} + \left.\frac{\partial^2\Psi}{\partial y^2}\right|_{i,j} \approx \frac{1}{h^2}\begin{array}{|c|c|c|}\hline & 1 & \\ \hline 1 & -4 & 1 \\ \hline & 1 & \\ \hline\end{array}\,\Psi_{i,j}. \tag{5.25}$$

The corresponding A matrix has a tri-banded structure, with diagonal sub-sections corresponding to the boundary values. The form of the A matrix is shown in Fig. 5.2 (we do not show the values of the entries of the matrix due to space considerations, but they can be obtained through the MATLAB symbolic codes available in the reproducible figure package of the dissertation, see Chapter 11). As in the one-dimensional case, all diagonal entries (which are the boundary value entries B_b) of A are eigenvalues, and the remaining eigenvalues are real and equivalent to those of a matrix A' which is real-symmetric, thus allowing the use of the bounds in Eq. (5.8). The form of A' is also shown in Fig. 5.2.

The unique forms of the Gershgorin disk centers and radii for A' are shown in Table 5.4. The stability bounds are once again the same as in Eq. (5.17) with $\vec{B}$ being the

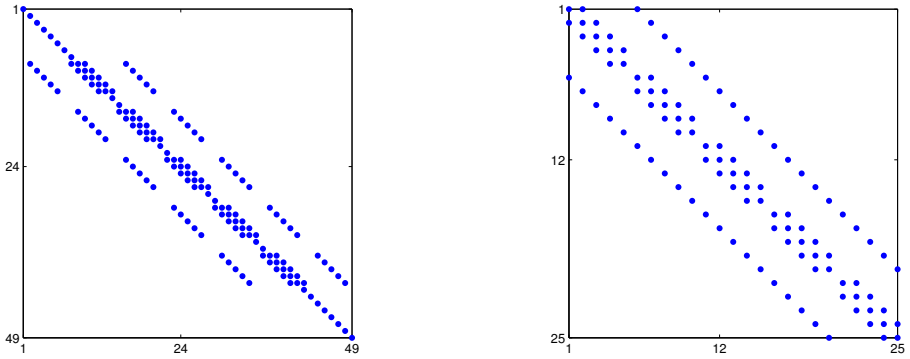


Figure 5.2. Form of scheme matrix A and A' for second-order central differencing of the two-dimensional NLSE. The dots represent non-zero entries of the matrices. The matrices shown are for a 7×7 grid.

Table 5.4. Gershgorin disk centers and radii for the A' matrix of the two-dimensional central-difference scheme.

a_{ii}	$r_i = \sum_{i \neq j} \lvert a_{ij} \rvert$
$L_i - 4$	2
$L_i - 4$	3
$L_i - 4$	4

set of all boundary values B_b and with $\vec{G}$ now defined as

$$\vec{G} = \{0, 1, 2, 6, 7, 8\}. \tag{5.26}$$

In the linear case ($s = 0$) with no external potential and periodic, Dirichlet or Laplacian-zero boundary conditions, we get the linear stability bound

$$k < \frac{\sqrt{8}}{8} \frac{h^2}{a}. \tag{5.27}$$

As before, since within the A matrix, all boundary, potential, and nonlinear terms are $O(h^2)$ the simple bound of Eq. (5.27) with slight adjustment can be used in many applications.

5.4.2 Fourth-Order Central Difference

The fourth-order central difference scheme in two dimensions is given by

$$\nabla^2\Psi_{i,j} \approx -\frac{1}{12\,h^2}\begin{array}{|c|c|c|c|c|}\hline & & 1 & & \\ \hline & & -16 & & \\ \hline 1 & -16 & 60 & -16 & 1 \\ \hline & & -16 & & \\ \hline & & 1 & & \\ \hline\end{array}\Psi_{i,j}. \tag{5.28}$$

The low-storage version of the 2SHOC equivalent scheme is defined in Eqs. (3.18) and (3.19) reproduced here:

$$1)\quad D_{i,j} = \frac{1}{h^2}\begin{array}{|c|c|c|}\hline & 1 & \\ \hline 1 & -4 & 1 \\ \hline & 1 & \\ \hline\end{array}\Psi_{i,j} \tag{5.29}$$

$$2)\quad \nabla^2\Psi_{i,j} \approx -\frac{1}{12}\begin{array}{|c|c|c|}\hline & 1 & \\ \hline 1 & -12 & 1 \\ \hline & 1 & \\ \hline\end{array} D_{i,j} + \frac{1}{6\,h^2}\begin{array}{|c|c|c|}\hline 1 & & 1 \\ \hline & -4 & \\ \hline 1 & & 1 \\ \hline\end{array}\Psi_{i,j}. \tag{5.30}$$

The corresponding A matrix has a five-banded structure. The structure of the A matrix and its corresponding A' matrix are shown in Fig. 5.3.

The resulting unique forms of the Gershgorin disk centers and radii for A' are shown in Table 5.5. The linearized stability bound is once again Eq. (5.17) but with $\vec{G}$ defined as

$$\vec{G} = \frac{1}{12} \times \{128, 127, 126, 110, 109, 92, 24, 9, 8, -6, -7, -8\}\,. \tag{5.31}$$

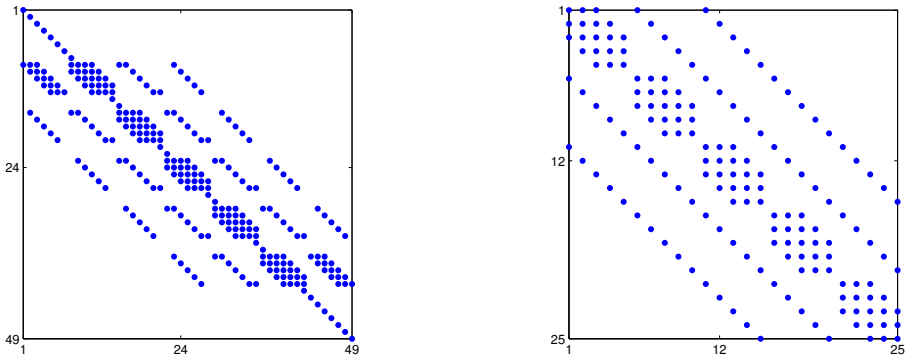


Figure 5.3. Form of scheme matrix A and A' for the fourth-order 2SHOC scheme of the two-dimensional NLSE. The dots represent non-zero entries of the matrices. The matrices shown are for a 7×7 grid.

Table 5.5. Gershgorin disk centers (a_{ii}) and radii (r_i) for the A' matrix of the two-dimensional 2SHOC scheme.

a_{ii}	$r_i = \sum_{i \neq j} \lvert a_{ij} \rvert$
$L_i - 5$	$11/2$
$L_i - 5$	$67/12$
$L_i - 5$	$17/3$
$L_i - 29/6$	$17/6$
$L_i - 59/12$	$25/6$
$L_i - 59/12$	$17/4$

The linear bound (with $s = 0$ and $V(\mathbf{r}) = 0$) is then given by

$$k < \frac{3\sqrt{8}}{32}\,\frac{h^2}{a}, \tag{5.32}$$

which, as in the one-dimensional case, is 25% lower than the second-order linear bound given in Eq. (5.27).

5.5 THREE-DIMENSIONAL STABILITY ANALYSIS

For the stability analysis in three dimensions, the same procedure utilized in the two-dimensional case of Sec. 5.4 is used.

5.5.1 Second-Order Central Difference

The second-order central difference scheme in three dimensions is given by

$$\begin{aligned}\nabla^2\Psi_{i,j,k} &= \left.\frac{\partial^2\Psi}{\partial x^2}\right|_{i,j,k} + \left.\frac{\partial^2\Psi}{\partial y^2}\right|_{i,j,k} + \left.\frac{\partial^2\Psi}{\partial z^2}\right|_{i,j,k} \\ &\approx \frac{1}{h^2}\left(\begin{array}{|c|c|c|}\hline & & \\ \hline & 1 & \\ \hline & & \\ \hline\end{array}\,\Psi_{i,j+1,k} + \begin{array}{|c|c|c|}\hline & 1 & \\ \hline 1 & -6 & 1 \\ \hline & 1 & \\ \hline\end{array}\,\Psi_{i,j,k} + \begin{array}{|c|c|c|}\hline & & \\ \hline & 1 & \\ \hline & & \\ \hline\end{array}\,\Psi_{i,j-1,k}\right),\end{aligned} \tag{5.33}$$

and the structure of the corresponding A and A' matrix are given in Fig. 5.4. The unique forms of the Gershgorin disk centers (a_{ii}) and radii (r_i) for A' are shown in Table 5.6. The

Table 5.6. Gershgorin disk centers (a_{ii}) and radii (r_i) for the A' matrix of the three-dimensional central difference scheme.

a_{ii}	$r_i = \sum_{i\neq j} \lvert a_{ij}\rvert$
$L_i - 6$	3
$L_i - 6$	4
$L_i - 6$	5
$L_i - 6$	6

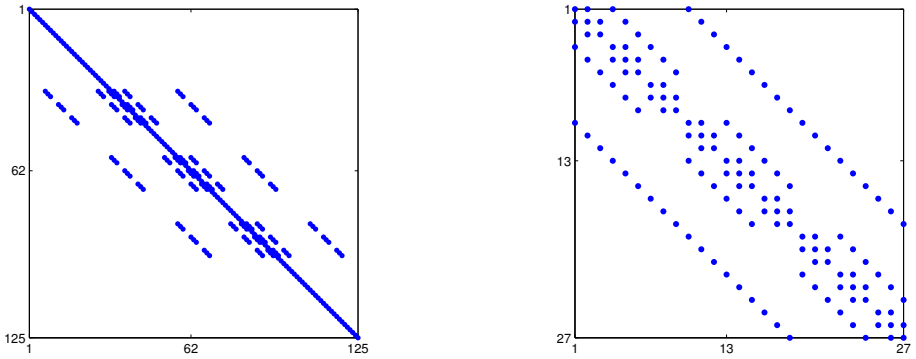


Figure 5.4. Form of scheme matrix A and A' for the second-order central difference scheme of the three-dimensional NLSE. The dots represent non-zero entries of the matrices. The matrices shown are for a 5×5 grid.

stability bounds of Eq. (5.17) in this case has $\vec{G}$ defined as

$$\vec{G} = \{12, 11, 10, 9, 3, 2, 1, 0\}. \tag{5.34}$$

In the linear case ($s = 0$) with no external potential and periodic, Dirichlet or Laplacian-zero boundary conditions, the linear stability bound becomes

$$k < \frac{\sqrt{8}}{12} \frac{h^2}{a}. \tag{5.35}$$

5.5.2 Fourth-Order Central Difference

The fourth-order central difference scheme in three dimensions is given by

$$\begin{aligned} \nabla^2 \Psi \approx \frac{1}{12\,h^2} \big[& \Psi_{i+2,j,k} + \Psi_{i-2,j,k} + \Psi_{i,j+2,k} + \Psi_{i,j-2,k} + \Psi_{i,j,k+2} + \Psi_{i,j,k-2} \\ & -16\,(\Psi_{i+1,j,k} + \Psi_{i-1,j,k} + \Psi_{i,j+1,k} + \Psi_{i,j-1,k} + \Psi_{i,j,k+1} + \Psi_{i,j,k-1}) + 90\,\Psi_{i,j,k} \big]. \end{aligned} \tag{5.36}$$

The single-storage version of the 2SHOC equivalent scheme is defined as in Eq. (3.26) and (3.27) and the structure of the corresponding A and A' matrix are given in Fig. 5.5, while the unique forms of the Gershgorin disk centers and radii for A' are shown in

Table 5.7. The stability bounds of Eq. (5.17) in this case has $\vec{G}$ defined as

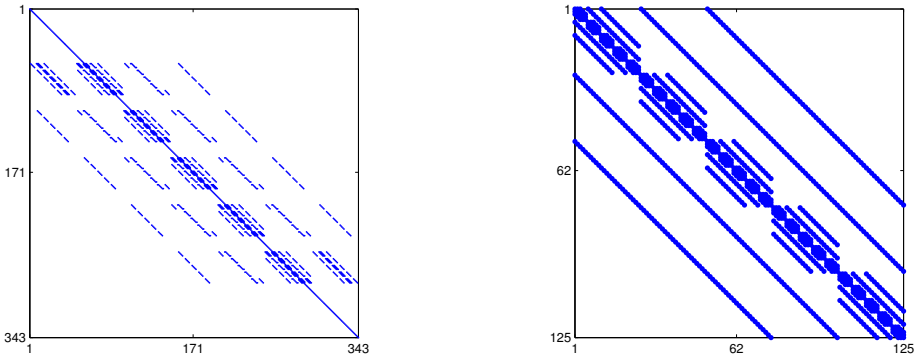


Figure 5.5. Form of scheme matrix A and A' for the fourth-order 2SHOC scheme of the three-dimensional NLSE. The dots represent non-zero entries of the matrices. The matrices shown are for a $7 \times 7 \times 7$ grid.

Table 5.7. Gershgorin disk centers (a_{ii}) and radii (r_i) for the A' matrix of the three-dimensional 2SHOC scheme.

a_{ii}	$r_i = \sum_{i \neq j} \lvert a_{ij} \rvert$
$L_i - 15/2$	$33/4$
$L_i - 15/2$	$25/3$
$L_i - 15/2$	$101/12$
$L_i - 15/2$	$17/2$
$L_i - 22/3$	$67/12$
$L_i - 22/3$	$17/3$
$L_i - 29/4$	$17/4$
$L_i - 89/12$	$83/12$
$L_i - 89/12$	7
$L_i - 89/12$	$85/12$

$$\vec{G} = \frac{1}{12} \times \{192, 191, 190, 189, 174, 173, 172, 156,$$
$$155, 138, 36, 21, 20, 6, 5, 4, -9, -10, -11, -12\} .$$

In the linear case ($s = 0$) with no external potential and periodic, Dirichlet or Laplacian-zero boundary conditions, the stability bound reduces to

$$k < \frac{\sqrt{8}}{16} \frac{h^2}{a}, \tag{5.37}$$

which, as in the one- and two-dimensional cases, is again only a 25% reduction than that of the second-order bound given in Eq. (5.35).

The stability bound results of this chapter are summarized in Appendix F for quick reference. The full research scripts of the NLSE magic software code package described in Chapter 8 can be set to compute the linearized stability bounds automatically. We have found through numerical testing that to ensure stability in all dimensions for typical problems, the bounds must be lowered by about 10%–20% (most likely due to nonlinear effects). Also, we note that the linearized results are often similar to the reduced linear bounds and can therefore be used as a good estimate of the true stability bound.

Now that the description of the numerical methods to be used in the current study are complete, we now discuss the code implementation of the methods.

CHAPTER 6

CODE IMPLEMENTATIONS:

MATLAB SCRIPT AND MEX NLSE INTEGRATORS

In the following three chapters, we describe in detail the code implementation of the numerical methods described in Chapters 3 and 4. The codes are combined into a MATLAB-based software package named 'NLSEmagic' which stands for "Nonlinear Schrödinger Equation Multidimensional MATLAB-based GPU-accelerated Integrators using Compact High-order Schemes". We leave the details of the NLSEmagic software package to Chapter 8 and first focus on the core of the program which are the NLSE integrators. NLSEmagic contains both serial and parallel versions of the integrators, the parallel integrators written in CUDA C to be run on GPUs. A detailed description of the GPU integrators will be given in Chapter 7. In the present chapter, we describe the implementation of the serial integrators, which include compiled C codes with a MEX interface which allows for efficient integration into a MATLAB-based code environment. To aid the discussion of the details of the C integrators, the source code of the three-dimensional CUDA C MEX integrator code in double precision is included in Appendix H.

The serial codes in NLSEmagic are included for two reasons. One, they allow calculations of the speedup of the CUDA codes and validate their output. Second, the MEX serial codes are quite usable on their own, as they are very simple to install/compile and run, and, as shown in Sec. 6.3.1, are much faster than their equivalent MATLAB script codes. Script integrators are also included as they are useful as a platform for testing new numerical schemes and boundary conditions.

All NLSE integrators in NLSEmagic are set-up to simulate a 'chunk' of time-steps of the NLSE and return the resulting solution Ψ. The 'chunk-size' is set based on the desired number of 'frames' (i.e. the number of times Ψ is accessed for plotting and analysis). For the serial integrators, this chunk-based approach is not necessary, but as we will see in Sec. 7.3.1, the chunk-based approach is very important for the CUDA integrators.

6.1 EXAMPLE TEST PROBLEMS

In this section, we describe the details of the example test problems that we will make use of in testing the speedup of the codes both in this chapter and in Chapter 7.

In one-dimension, the exact co-moving dark soliton solution given by Eq. (2.5) in Chapter 2 is used with the parameters $s = -1$, $a = 1$, $c = 0.5$, and $\Omega = -1$. The spatial grid-size is set to $h = 0.1$. The computed linear and linearized stability bounds of Chapter 5 for the RK4+CD scheme yield a maximum available time-step of $\mathbf{k} = 0.0070711$ and $\mathbf{k} = 0.0070534$ respectively. For the RK4+2SHOC scheme, the stability bounds are $\mathbf{k} = 0.0053033$ and $\mathbf{k} = 0.0052934$ respectively. Therefore, we use a time-step of $\mathbf{k} = 0.005$ for all simulations to ensure stability and consistency. The soliton solution within our simulations is shown in Fig. 6.1. Since the solution has an approximate constant modulus-square value at the boundaries we utilize the MSD boundary condition described in Chapter 4.

In two dimensions, we use an approximation to a steady-state dark vortex solution of charge $m = 1$ to the NLSE with $V(x) = 0$ and $s < 0$ given by Eq. (2.16) in Chapter 2. Since we will be simulating the vortex solution on very large grids (up to nearly 2000×2000), inserting the interpolated exact numeric solution of the vortex profile $f(r)$ onto the two-dimensional grid can take considerable time. Because of this, and since we are not interested in the exact solutions themselves, but rather are using them to have non-trivial test problems, we do not use the numerically exact vortex profile and instead

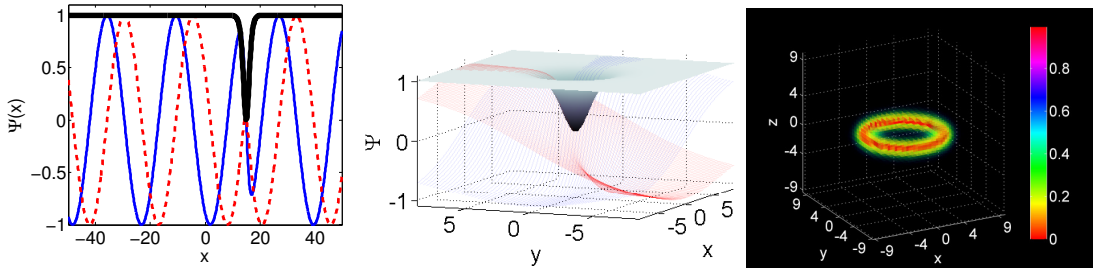


Figure 6.1. Left: Example of the one-dimensional dark soliton of Eq. 2.5 at time $t = 30$. The thin solid (blue) line and the thin dashed (red) line are the real and imaginary parts of Ψ respectively. The thick solid (black) line is the modulus-squared of the wave-function $|\Psi|^2$. The grid size is $\mathbf{N} = 1000$. Middle: Example of the approximate two-dimensional dark vortex of Eq. (2.16) at time $t = 0$. The (blue) and (red) mesh lines are the real and imaginary parts of Ψ respectively. The solid (gray) surface is the modulus-squared of the wave-function $|\Psi|^2$. The grid-size is $\mathbf{N} = 70 \times 70$. Right: Example of the approximate three-dimensional dark vortex ring of Eq. (2.45) at time $t = 0$. The image shows an inverted-volumetric rendering of the modulus-squared of Ψ. The grid size is $\mathbf{N} = 29 \times 29 \times 29$.

use an approximation to it given by the one-dimensional dark soliton of Eq. (2.5) with $c = 0$ with $x > 0$.

The spatial step-size is chosen to be $h = 0.25$ which allows for a maximum stable time-step of approximately $\mathbf{k} = 0.016$. However, in order to allow for better comparisons between simulations of different dimensionality, we set the time-step to be equal to that of the one-dimensional examples, $\mathbf{k} = 0.005$. A depiction of the approximate dark vortex within our simulations is shown in Fig. 6.1.

In three dimensions, we use an approximation to a dark vortex ring (of charge $m = 1$) solution to the NLSE amidst a co-moving back-flow which causes the vortex ring to be a steady-state solution. The form of the initial condition is given by Eq. (2.45) where $g(r, z)$ is chosen to be the numerically-exact two-dimensional $m = 1$ dark vortex (which, since we only are using two-dimensional interpolations of resolutions up to 144×144, is able to be formulated efficiently in this case) at position $r = d$ in the r–z half-plane, where d is the radius of the vortex ring. The steady-state vortex ring is not numerically optimized (as will be done in Chapter 9) in order to reduce the time it takes to formulate the initial condition. In our example, we choose $d = 5$ and use a spatial step-size of $h = 2/3$. Once

again, for comparison purposes, we use a time-step of $\mathbf{k} = 0.005$ even though the scheme would be stable with a larger time-step (as large as 0.045). A depiction of the vortex ring within our simulations is shown in Fig. 6.1.

6.2 MATLAB SCRIPT CODE INTEGRATORS

The script implementations of the NLSE integrators are used to quickly develop and test new methods, and for comparison of the speedup of the MEX integrators. They are split into two script files. The main driver scripts (`NLSExD.m`) compute the chunk of time-steps in a `for` loop, while the computations of $F(\Psi)$ with the desired boundary conditions in the RK4 scheme of Eq. (3.1) are done by calling a separate script file (`NLSExD_F.m`). All of the scripts utilize MATLAB vectorized operations wherever possible for efficiency [i.e. using `A+B;` instead of `for i=1:N; A(i)+B(i); end;`].

The script integrator are very straight-forward implementations of the numerical schemes. They have been validated by numerous simulations of known solutions to the NLSE and LSE.

6.3 MATLAB MEX CODE INTEGRATORS

The NLSEmagic serial MEX codes are stored in C source code files and integrate the NLSE for a given number of time steps and return the result to MATLAB.

When using a MEX file, it is important to handle complex values correctly. Since MEX files are written in C, one must either use a complex-number structure, or split the real and imaginary parts of the variables into separate vectors and write the numerical schemes accordingly. As shown in Ref. [78], for CUDA codes, using complex structures is less efficient than splitting the real and imaginary parts directly. To keep the serial and GPU codes as similar to each other as possible, the split version of the numerical schemes

are used. Writing $F(\Psi)$ in this way yields

$$F(\Psi)^R_{i,j,k} = -a\,\nabla^2\Psi^I_{i,j,k} - s\left[\left(\Psi^R_{i,j,k}\right)^2 + \left(\Psi^I_{i,j,k}\right)^2\right]\,\Psi^I_{i,j,k} + V_{i,j,k}\,\Psi^I_{i,j,k} \qquad (6.1)$$
$$F(\Psi)^I_{i,j,k} = \;\; a\,\nabla^2\Psi^R_{i,j,k} + s\left[\left(\Psi^R_{i,j,k}\right)^2 + \left(\Psi^I_{i,j,k}\right)^2\right]\,\Psi^R_{i,j,k} - V_{i,j,k}\,\Psi^R_{i,j,k},$$

where Ψ^R and Ψ^I are the real and imaginary parts of the solution respectively. Additional operations are straight-forward (for the split version of the MSD boundary condition, see Chapter 4).

A MEX code in C is very similar to a regular C code, but uses MATLAB-specific interfaces for input and output data, and special MATLAB versions of memory allocation functions. Instead of having a `main()` function as in standard C, the primary program is within a function called `mexFunction(nlhs, plhs[], nrhs, prhs[])`. The pointer `prhs[]` (pointer-right-hand-side) gives the C code access to the input data being sent from MATLAB, where `nrhs` is the number of input variables (including arrays and scalars). The pointer `plhs[]` (pointer-left-hand-side) is used in allocating memory for output arrays and scalars, allowing the MEX code to return data to MATLAB.

Special MEX functions (`mxGetPr()`, `mxGetPi()` and `mxGetScalar()`) are used to extract the data from the `prhs[]` array of MATLAB inputs. The function `mxGetScalar()` is used to extract a scalar value while the functions `mxGetPr()` and `mxGetPi()` retrieve the pointer to the real and imaginary part of the selected input array respectively. If the input array is fully real (which can be often the case, especially for initial conditions), then the `mxGetPi()` is null-valued which causes a segmentation fault if accessed. To get around this problem, code has been added that checks to see if there is an imaginary part of the input array of Ψ (using the function `mxIsComplex()`), and if not, allocates an all-zero imaginary array manually (which is then freed at the end of the MEX code).

To allocate memory space for output arrays, the MEX functions `mxCreateDoubleMatrix()` (for 1D and 2D arrays) and `mxCreateNumericArray()` (for

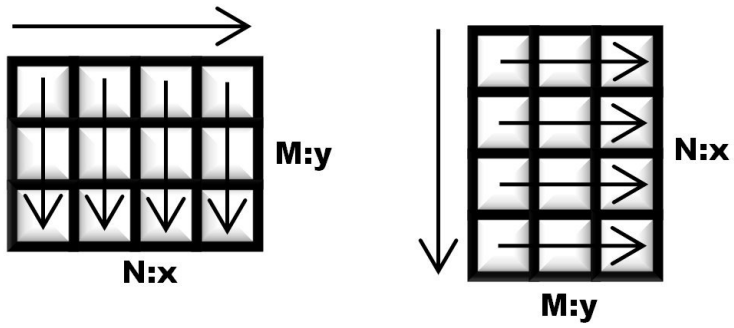


Figure 6.2. Representation of a two-dimensional array in MATLAB (left) and in C (MEX) (right). The arrows indicate the linear access pattern. The x and y dimensions shown are the x and y dimensions of the solution Ψ.

higher dimensionality arrays) are used to allocate memory space which is then pointed to by `plhs[i]`. This memory is then seen as a one-dimensional array by the C code, but in MATLAB will be the desired dimensionality.

In order to use the input and output arrays in C, proper indexing is essential. The MEX functions `mxGetN()` and `mxGetM()` are used to get the dimensions of one-dimensional and two-dimensional arrays (see below for three-dimensional array handling). However, because MATLAB stores arrays column-order, while C stores them row-order, the M value obtained from `mxGetM()` in the C code is actually the number of columns, while the N value obtained from `mxGetN()` is the number of rows (see Fig. 6.2). Therefore, the input array is seen by the MEX file as transposed from the way it is seen in MATLAB (of course no actual transpose operation is performed in the MATLAB to C passing of arrays, only the order in which the different languages access the elements is different). Thus, while in MATLAB, the column length of the Ψ solution array is the number of grid points in the x-direction of Ψ and the row length is the number in the y-direction, inside the MEX file, this is reversed. (As will be seen in Sec. 7.2, this makes the 'x' CUDA grid dimension to be along the y-direction of the solution and vice-versa). The difference between the row-order access of C and the column-order access of MATLAB must be taken into account to correctly access the elements in an input array

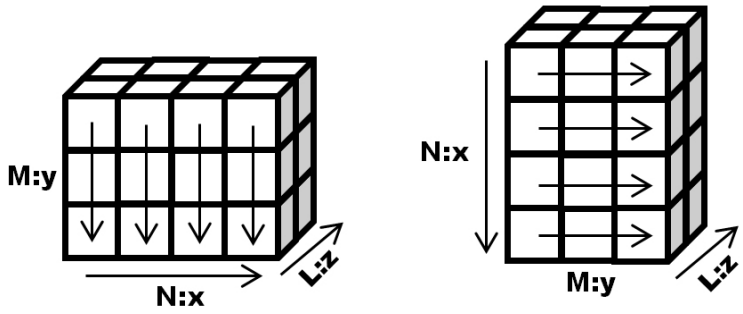


Figure 6.3. Representation of a three-dimensional array in MATLAB (left) and in C (MEX) (right). The arrows indicate the linear access pattern. The x, y and z dimensions shown are the x, y and z dimensions of the solution Ψ. Here, the z dimension maintains its orientation in the access pattern, while the x and y dimensions are transposed.

(for our discussion here, denoted A). To access array element (i, j) (where $i \in [0, N-1]$ and $j \in [0, M-1]$) of array A in the MEX file, one uses `A[M*i + j]`, where M is the number of columns obtained from `mxGetM()`.

In three dimensions, the MEX functions `mxGetNumberOfDimensions()` and its counterpart `mxGetDimensions()` are used to get the dimensions of the three-dimensional input arrays. The `mxGetDimensions()` returns an array of dimension lengths which, in our codes, are stored as L, M, and N. As in the two-dimensional case, the access pattern differs in MATLAB and C. Once again, the M and N dimensions of the MATLAB array are seen by the MEX file as swapped (see Fig. 6.3). Thus, in Sec. 7.2, the CUDA grid dimensions for x will be along the y-direction and vice-versa but the z grid dimension will correspond to the z-direction of the solution. To access array element (i, j, k) (where $i \in [0, L-1]$, $j \in [0, N-1]$, and $k \in [0, M-1]$) of array A in the MEX file, one uses `A[M*N*i + M*j + k]`.

Within a MEX file, memory allocations and deallocations are performed using MEX comparable functions to the C functions `malloc()` and `free()` called `mxMalloc()` and `mxFree()`. The MEX function `mxMalloc()` allocates the desired memory in a MATLAB memory-managed heap. The advantage of using `mxMalloc()` instead of

`malloc()` is that if a MEX file fails, or is interrupted during its execution, MATLAB can handle and free the memory to better recover from the failure [108].

The compilation of a MEX file is performed within MATLAB using the `mex` command as `mex mymexcode.c`. The MEX compiler uses whatever C compiler on the current machine is selected by MATLAB. By default, MATLAB uses the included LCC compiler, but this can be changed using the `mex -setup` command in MATLAB. For the CUDA MEX codes on Windows, the compiler must be set to Microsoft Visual C++ while on Linux, the standard GCC compiler can be used. Thus for our serial MEX codes, we also use these compiler options. After a MEX file is compiled, the resulting binary file has a special extension which depends on the operating system being used (`.mexw32` and `.mexw64` for 32-bit and 64-bit Windows respectively, and `.mexglx` and `.mexa64` for 32-bit and 64-bit Linux respectively).

Both single and double precision versions of the NLSEmagic serial MEX integrator codes have been developed in order to properly compare the single-precision versions of the CUDA MEX codes to their serial counterparts (see Sec. 7.3). The single precision MEX codes convert the input arrays using the cast `(float)`, and then convert the output arrays back to double precision (MATLAB's native format) after the computation of the desire number of time-steps.

6.3.1 Speedup of Serial MEX Codes

To illustrate the advantage of using MEX codes over script codes in MATLAB to integrate the NLSE, we show timing results comparing the performance of the script integrators of NLSEmagic to the equivalent MEX codes using the example problems described in Sec. 6.1. As mention in Sec. 6.2, our script codes are sufficiently optimized by utilizing MATLAB's built-in vector operations wherever possible. We set an end-time of $t = 5$, the number of frames to 10, and use double precision for all simulations as it is MATLAB's native format for the script codes. The simulations are performed for a total number of grid points of $\mathbf{N} = 10000$ to $\mathbf{N} = 1000000$. The results are shown in Fig. 6.4.

We see that using MEX codes are, on average, eight to fifteen times faster than the equivalent script codes. Although there is variation in the speedup (more for lower dimensions and variable over problem size), these variations are small and therefore the speedup of the MEX integrators can be considered approximately constant for all dimensions and problem sizes. Therefore, the serial MEX codes included in NLSEmagic are valuable in their own right (especially for researchers who are currently using script codes) since no special setup is needed to compile them and their use is straight-forward.

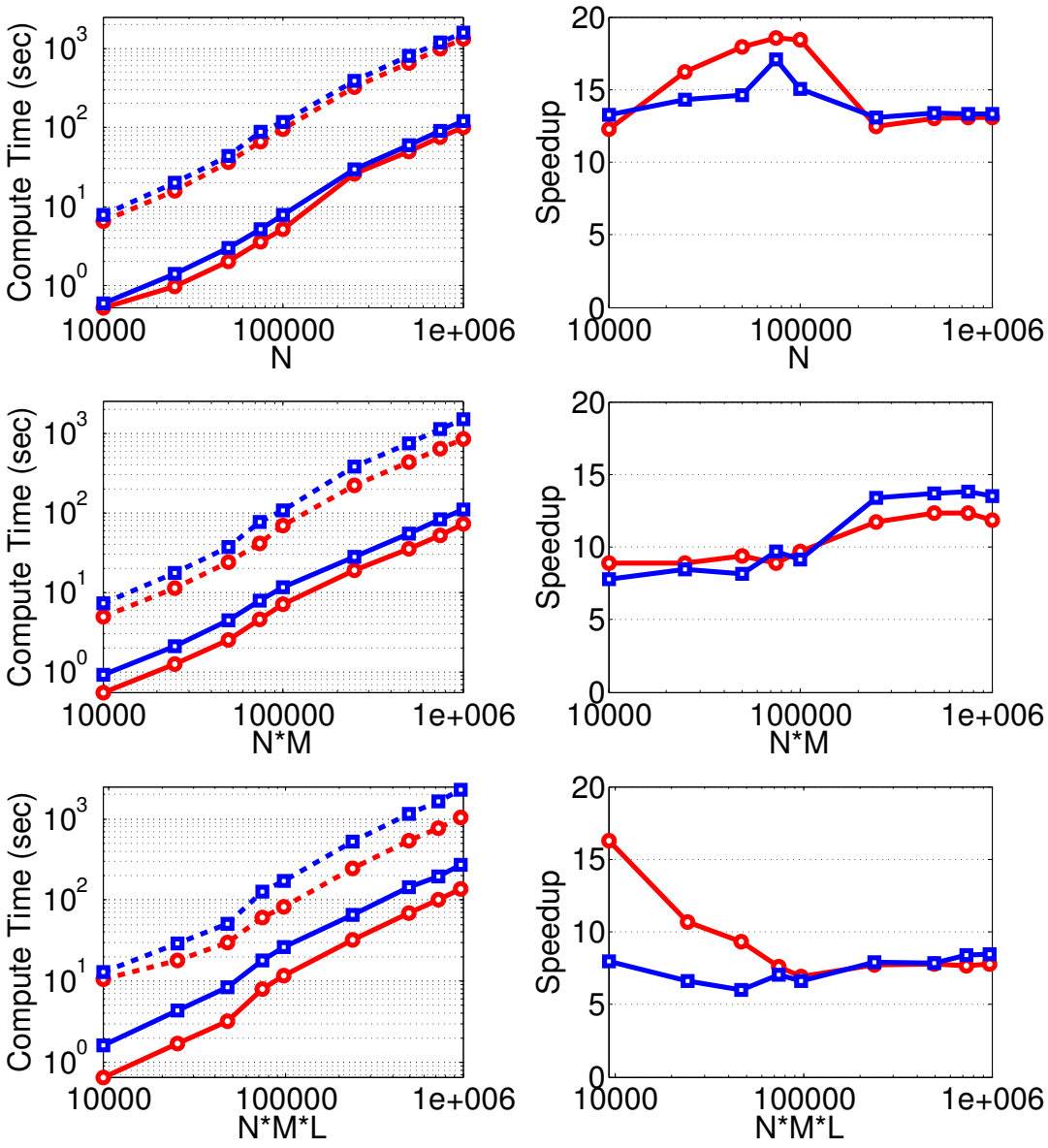


Figure 6.4. Timing results comparing script (dashed lines) and C MEX (solid lines) NLSE double-precision integrators for both the RK4+CD (red circles) and RK4+2SHOC (blue squares) schemes using the example problems of Sec. 6.1 with an end-time of $t = 5$ and a number of frames of 10. The simulations are performed for a total number of grid points of $\mathrm{N} = 10000$ to $\mathrm{N} = 1000000$. The results for the one-, two-, and three-dimensional codes are shown top to bottom. The left column displays the simulation times and the right column shows the corresponding speedup of the MEX codes.

CHAPTER 7

CODE IMPLEMENTATIONS:

GPU-ACCELERATED CUDA NLSE INTEGRATORS

In this chapter, we detail the implementation of the GPU-accelerated NLSE integrators in NLSEmagic and test their performance. In order to familiarize the reader to GPU programing, an overview of GPU computing including the structure of the hardware, programming APIs, compatibility, and portability is provided.

7.1 General Purpose GPU Computing with NVIDIA GPUs

Although there are other companies that produce GPU graphics cards (such as ATI), NVIDIA is by far the most established when it comes to general-purpose GPUs (GPGPU). Their compute-only Tesla GPGPU cards as well as their native compute unified device architecture (CUDA) API and enormous support and development infrastructure make NVIDIA the most logical choice for developing and running GPU codes (compatibility and portability concerns will be discussed in Sec. 7.1.4). In this section we focus on the physical structure of the GPUs as well as the logical structure defined by the CUDA API.

7.1.1 NVIDIA GPU Physical Structure

Since the programming model for NVIDIA GPUs is very closely related to their physical design, it is important to have an overview of their physical structure. Also, because NVIDIA produces several different models and architectures, a description of these varieties is important for proper implementation of GPU codes.

A typical GPU is a separate piece of hardware that is connected to a PC through a PCIe slot on the motherboard. However, NVIDIA also makes built-in GPU motherboards,

as well as integrated GPUs for laptop computers. A simplified schematic of a current-model (Fermi architecture) NVIDIA GPU is shown in Fig. 7.1. The GPU contains

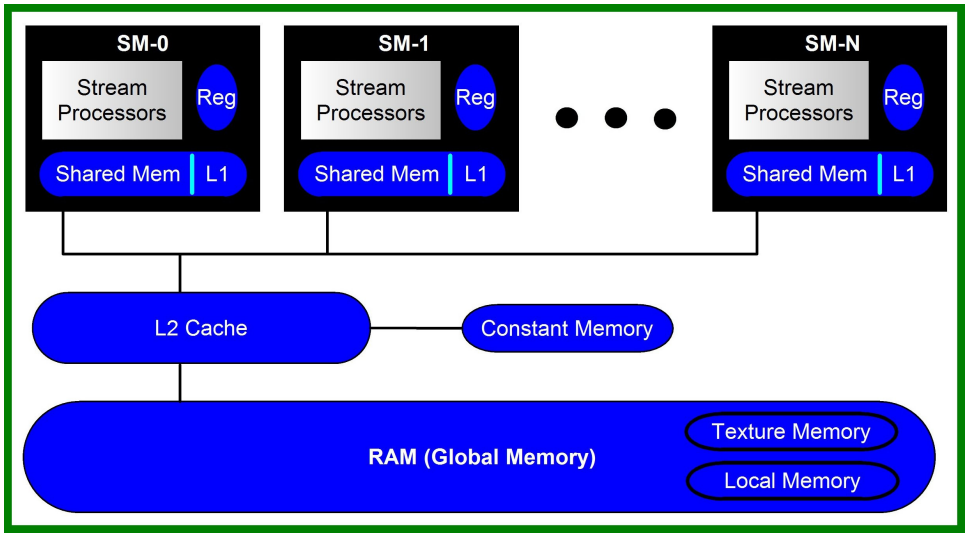


Figure 7.1. Simplified schematic of an NVIDIA GPU with Fermi architecture (A Tesla architecture looks the same but without caches).

a number of stream multiprocessors (SM) which each contain a number of stream processors (SP) which are the compute cores of the GPU. Each SM performs computations independently and simultaneously from the other SMs and has no connectivity or communication with other SMs. Each SM has a small, fast memory which is configured to allocate some space for a level 1 (L1) cache and the rest of the space for shared memory. The shared memory is shared between all SPs, so each core can access any part of shared memory. Each SM also has fast registers for the SPs to use for computations and storage of local variables.

The main memory of the GPU is a large amount of DRAM called the *global memory space* which, depending on the GPU, can have a capacity from 512MB up to 12GB. Accesses to the global memory from the SPs are (reportedly) two orders of magnitude slower than accesses to shared or constant memory (an additional small [64KB] globally-accessible memory space which is as fast as shared memory) [109]. Parts

of the global memory space are used for local variables (when they cannot fit into registers or cache) and texture memory which is mainly only allocated for graphics processing. Between the DRAM and the SMs is a level 2 (L2) cache which improves memory performance when SMs access global memory. Additional hardware details are beyond the required scope of this dissertation.

A typical GPU code transfers data from the host computer's RAM to the GPU RAM, performs computations using the data on the SMs of the GPU, and then when completed, transfers the resulting data back to the host computer's RAM. The memory transfer has large latency associated with it, and so it is ideal to compute as much as possible on the GPU before transferring the data back. However, since the host computer cannot 'see' the data until it has been transfered back from the GPU, a trade-off between maximum performance and usability is often encountered. However, for many applications (including NLSEmagic), this issue is easily resolved (see Sec. 7.3.1).

An additional issue to keep in mind is that if one is using a video-enabled GPU, there can be a run-time limit on individual GPU function calls. Each call to an executable GPU routine cannot take more than the display driver timeout limit of the operating system (typically a few seconds) to compute, otherwise the display driver will shut down and reset. This is usually not a problem because most GPU codes (including NLSEmagic) run many small GPU function calls, each of which take less than the run-time limit to compute. However, if the run-time limit does become a problem, one can simply run the codes on a compute-only GPU.

The first general purpose GPUs developed by NVIDIA utilized a chipset design called the *Tesla architecture*. The design was similar to that of Fig. 7.1, except that there were no caches. Each SM had 8 SPs and 16KB of shared memory. The arithmetic logic units (ALUs) of the SPs were not fully IEEE compliant which meant that results computed on the GPU could differ slightly from those computed on a CPU. Double-precision

arithmetic was not supported in early models, but was later added (again, not fully IEEE compliant). Also, the global memory RAM did not have ECC error correction.

The newest generation of NVIDIA GPUs implement a chipset called the *Fermi architecture*. The Fermi-based cards are a large improvement over the older Tesla-based cards. Some of the key improvements are [110]:

- Addition of an L1 and L2 cache system which helps reduce memory latency.
- ECC error-correcting memory (not available on GeForce model cards, see below).
- More SPs (cores) per SM (24-32), and more total SPs possible (up to 1024).
- Fully IEEE compliant floating-point operations.
- Amount of shared memory per SM increased to up to 48KB (instead of 16KB).
- Amount of compute threads increased from 512 to 1024 threads per block (see Sec. 7.1.2).

NVIDIA produces three GPU product lines which now all use the Fermi architecture. These are GeForce, Quadro, and Tesla (not to be confused with the Tesla *architecture*). The GeForce GPUs are graphics cards targeted towards gamers, the Quadro GPUs are graphics cards targeted towards professional graphic workstations, and the Tesla line of GPUs are compute-only devices with no display output targeted for scientific applications. Both the Quadro and Tesla cards are much more expensive than the comparable GeForce cards.

The basic internal chipset of the GeForce cards are the same as the Tesla cards with a few important differences [111]. The double precision arithmetic performance has been artificially reduced to 25% of the Tesla cards, and the ECC error correction is disabled (and cannot be enabled). However, the ECC memory option reduces performance and available memory when activated, and in many simulations, the order of accuracy is not sensitive enough to be effected by the lack of ECC memory. GeForce GPUs also tend

to come with less global RAM than the Tesla or Quadro cards. In addition, the Tesla cards have two copy engines, while the GeForce cards have one. This means that CUDA codes run on the Tesla cards can simultaneously compute and transfer memory to the CPU which increases performance. Tesla cards are also built with higher standards than GeForce cards, and are put through a zero-error tolerance stress test and burn-in. For all these reasons, Tesla cards are more desirable than GeForce cards for scientific codes. However, for moderate problems, the GeForce's abilities are often more than adequate, and their low price-tag makes them an affordable option for high-performance computing.

The current (as of this writing) top-of-the-line Fermi-based card for home PC use is the GeForce GTX 690 which has $3,072$ compute cores and 4GB RAM and costs approximately $1000. The current top Tesla card is the M2090 with 512 cores and 6GB memory and costs around $3000, while the current top-of-the-line Quadro card is the Quadro Plex 7000 with $1,024$ cores and 12GB RAM and sells for around $14,000.

7.1.2 CUDA API and Logical Structure

NVIDIA allows programmers to utilize its GPUs through an API called the compute unified device architecture (CUDA) API. CUDA is a code extension to C/C++ which gives low-level access to the GPUs memory and processing abilities. The CUDA codes are compiled by a provided compiler called `nvcc`.

There are two sections of a CUDA C code (usually written in a file with a `.cu` extension). One section of code is the host code which is executed on the CPU. This section contains setup commands for the GPU, data transfer commands to send data from the CPU to the GPU (and vice-versa), and code to launch computations on the GPU. The second part of a CUDA code contains what are known as CUDA *kernel* routines. These are routines which get compiled into binaries which are able to be executed on the GPU by calls invoked by the host code.

The CUDA programming model is based on a logical hierarchical structure of *grids*, *blocks*, and *threads*. This hierarchy is shown in Fig. 7.2. When a kernel is launched,

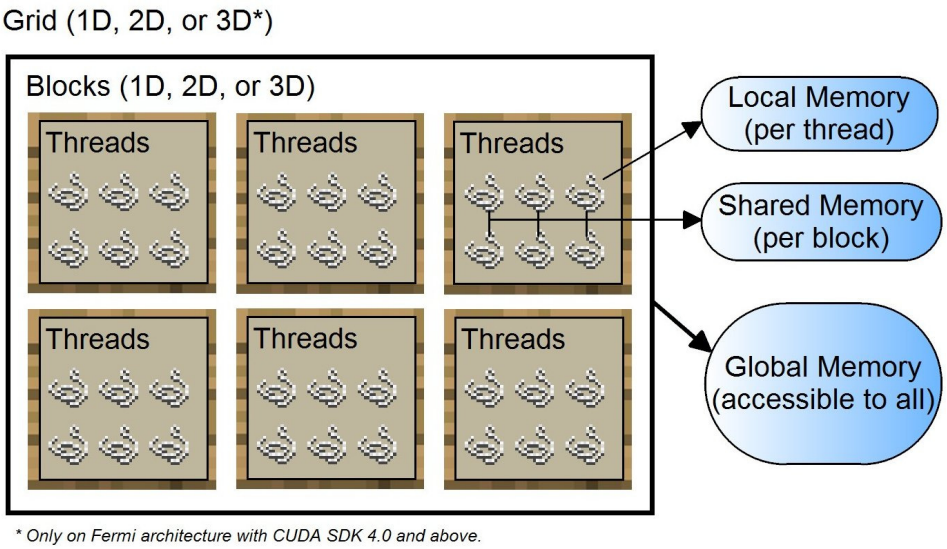


Figure 7.2. Schematic of the CUDA API logic structure.

it is launched on a user-defined compute-grid which can be one-, two-, or three-dimensional (three-dimensional compute-grids are only available on Fermi architecture with the CUDA SDK 4.0 or above installed). This grid contains a number of blocks where the computations to be performed are distributed. Each block must be able to be computed independently as no synchronization is possible between blocks. Each block contains a number of threads, which are laid out in a configuration that is one-, two-, or three-dimensional. The threads perform the computations and multiple threads are computed simultaneously on the SM. Synchronization of threads within a block is possible, but for best performance, should be kept to a minimum. Each thread has its own local memory and all threads in a block have access to a block-wide shared memory. All threads in all blocks on the grid have access to the global memory.

Each thread has access to intrinsic variables (stored in registers) which are used to identify its position in the compute-grid and block. The grid dimensions are stored in variables called `gridDim.x`, `gridDim.y`, and `gridDim.z` and the block dimensions are stored in `blockDim.x`, `blockDim.y`, and `blockDim.z`. The block that a thread is contained in is given in coordinates by the variables `blockIdx.x`, `blockIdx.y`, and (for

Fermi cards with SDK 4.0 and above) `blockIdx.z`. The thread's position in the block is given by the variables `threadIdx.x`, `threadIdx.y`, and `threadIdx.z`. All these variables are used to determine what part of the problem the specific thread should compute.

The logical structure relates directly to how the hardware of the GPU executes a kernel code. Each block is executed by an SM, where each thread is executed by an SP. This is why the threads within a block have access to a shared memory, but blocks cannot access each others shared memory. The local memory of each thread is stored in the registers of the SM, unless the register space is used up in which case the local variables get stored in cache, and, if the cache's are full, to global memory (reducing performance).

7.1.3 Compatibility

Since new and updated GPU cards are being developed and released continuously, compatibility can become a major concern. To help with this issue, NVIDIA has maintained a compatibility scheme known as the major/minor compute capability of a GPU. This is a number in the form of $y.x$ where y and x are the major and minor compute capabilities respectively. Whether code written and compiled for one GPU will work on a different model GPU depends on the compute capabilities of the two cards and on how the code was compiled.

CUDA codes can be compiled into a binary executable for a specific GPU known as 'cubin' objects, and/or be compiled into parallel thread execution (PTX) assembly code which is compiled into binary at runtime for the GPU being used. Each compilation scheme has separate compatibility rules.

Codes compiled into a cubin object are only compatible with future minor compute capability revisions. Therefore, for example, a cubin file compiled for compute capability 1.2 will run on CUDA cards with compute capabilities 1.2 and 1.3, but not with those of 1.1 or 2.0. In contrast, codes compiled into PTX assembly code are compatible with *any* future compute capability. However, features that are present in higher compute

capabilities cannot be used by previous PTX codes. For example, double precision is only supported in GPUs with compute capabilities of 1.3 and above. Therefore, if one has CUDA code which uses double precision, they must compile the PTX codes using compute capability 1.3 or above. If not, (say, compiling for PTX code of 1.0), the double precision gets demoted to single precision. A drawback of using PTX codes is that they increase the load time of the application due to the in-time compilation (although typically after the first compile, the binary is stored in a cache for future runs of the application) [109, 112].

The compiler options for the NLSEmagic code package (see Chapter 8) have been set to compile PTX code for 1.0, 1.3, and 2.0, as well as cubin objects for compute capability 2.0 (the major compute capability 2.x is Fermi architecture, while 1.x is Tesla). This ensures that the current form (Fermi-optimized) codes can compile (trying to compile to cubin files for old architecture fails when using new features such as larger shared memory and double precision).

To use NLSEmagic with older GPUs (lower than 2.0 compute capability), one must make minor modifications to some of the CUDA integrator codes and recompile them. The block sizes (in the .cu files as well as the driver scripts) must be altered so that there are less than 512 threads per block, and that the total shared memory per block is less than 16KB. Also, the double precision codes will not work with GPUs with compute capabilities less than 1.3. Additionally, for efficiency, the `nvmexopts(64).bat` or `nvopts.sh` (see Appendix I) should be altered to compile native cubin objects for the specified architectures compute capability level in order to avoid the loading delay of the in-time compilation of PTX codes.

7.1.4 Portability

A major reason many scientific programmers are hesitant to write their programs for GPUs and using CUDA to do so, is the issue of portability. The concern is that one could spend years developing GPU codes with CUDA only to find out that the GPUs have

been replaced with alternate technology and are no longer supported, or, that another company has overtaken the GPU market and NVIDIA's native CUDA API is no longer being supported.

As a solution to these concerns, a new API called OpenCL [113] has been developed by the Khronos Group [114] which can function across multiple GPU vendors, as well as other CPU architectures. Although code written in OpenCL is guaranteed to work on all multi-core architectures, specific optimization is necessary for the codes to work efficiently. A number of studies have been done comparing the performance of OpenCL versus CUDA [115–117]. In general, the CUDA codes outperform the OpenCL equivalent codes (sometimes by a significant percentage). However, with further OpenCL-specific performance tunings, it is possible to get the OpenCL codes to run nearly as fast as the CUDA codes. OpenCL and CUDA are very similar in terms of logical structure and code design. Therefore, codes written in CUDA can easily be changed into OpenCL codes (in fact, an *automatic* CUDA-to-OpenCL converter called `cu2cl` has recently been developed [118]).

Another option for portability is the Portland Group's new x86 CUDA compiler [119]. This allows the compilation of CUDA codes into binaries that are able to be run on multi-core x86 CPUs (with 64-bit support being developed). This will allow CUDA codes to be useful even if GPUs are overtaken by multi-core CPUs in the market. Similar to the x86 compiler is the MCUDA compiler created by the University of Illinois at Urbana-Champaign Center for Reliable and High-Performance Computing [120]. This compiler can take CUDA code and automatically convert it into pure C code that interfaces to a runtime library for parallel execution on multi-core PCs through the use of POSIX threads.

Another portability option is the Ocelot project from Georgia Tech [121] which allows CUDA codes that have been compiled into PTX code to be run on AMD GPUs as

well as CPU architectures without needing to recompile the codes. This would directly allow one to continue using CUDA codes on alternative hardware.

For developers who are still unsure about putting the time in to make CUDA codes, NVIDIA has recently announced a new compiler developed by the Portland Group which can compile a previously written serial C code and, with a few helper-comments, compile the code to run on a GPU automatically with no direct CUDA programming [122]. Although it is expected that the resulting code will exhibit much less speedup than directly coding the CUDA by hand, NVIDIA has guaranteed at least a 2X speedup when using the new compiler.

From the numerous portability options, we conclude that there is little risk in taking the time to develop GPU codes using CUDA, and the potential speedup of the codes far outweighs the risk.

7.2 GPU-Accelerated CUDA MEX Code Integrators

In this section we describe in detail the design of the CUDA MEX NLSE integrators in NLSEmagic. Both single and double precision versions of all integrators are produced in order for the codes to be compatible with older GPU cards, as well as to increase performance when double precision accuracy is not required (as will be shown in Sec. 7.3, the single precision GPU codes can be much faster than the double precision codes).

As mentioned in Sec. 7.1.2, there are two main sections of a CUDA code, the host code and the kernel codes. The host code is run on the CPU and is used to set up the problem, handle memory allocations and transfers to the GPU, and run the GPU kernel code. The kernels are the routines that are run on the GPU.

Since the native language for GPU-compatible CUDA codes is C, adding GPU functionality to MATLAB is in theory straightforward by using the MEX compiler interface (however setup can be difficult, an issue addressed in Chapter 8). The newest versions of MATLAB have GPU-compatibility built-in [123], and ways to compile CUDA

C code segments into callable MATLAB functions. However, because many researchers do not have the newest versions of MATLAB, in the interest of portability, it is preferable to not use these built-in functions, but rather write C codes which contain CUDA code directly, and compile them with a CUDA-capable MEX compiler called NVMEX [124].

A MATLAB-interfaced CUDA MEX code is almost identical to a standard CUDA C code except that its host code uses the MEX-specific functions described in Sec. 6.3 for input, output, and CPU memory allocation. However, since the kernel code is independent of the MEX interface, they are identical to kernels one would write for a standard CUDA C code, and therefore can be easily ported if desired.

The GPU-accelerated MEX routines in NLSEmagic are called from MATLAB in the exact same way as the serial MEX routines described in Sec. 6.3. That is, they have the same input and output parameters, and they integrate the NLSE over a specified number of time-steps (the chunk-size) before returning the current solution Ψ to MATLAB.

The basic outline of the integrators is that the input solution Ψ and potential $V(\mathbf{r})$ arrays obtained through the MEX interface are transfered to the GPU's memory. Then, the desired number of time-steps are computed on the GPU using kernel functions, and the resulting solution is transfered back to the CPU memory, where it is outputted to MATLAB through the MEX interface.

Since the memory transfers from CPU to GPU and vice-versa are quite slow, the larger the number of time-steps the GPU computes before sending the solution back to the CPU (i.e. the larger the chunk-size), the better the expected speedup. In Sec. 7.3.1 we investigate this in detail for our simulations.

7.2.1 Specific Code Design for all Dimensions

We now discuss the details of the CUDA implementation which are the same for the one-, two-, and three-dimensional integrator codes. In the sections that follow, we will discuss dimension-specific code design as well. We focus on the CUDA design aspects of the codes, and therefore do not mention any MEX interfaces (as they are all equivalent to

the serial MEX codes previously described). Thus, we assume that the solution Ψ and potential $V(\mathbf{r})$ have already been obtained through the MEX interface along with all scalar parameters. As mentioned in Chapter 6, the three-dimensional CUDA MEX integrator source code for the 2SHOC scheme in double precision is provided in Appendix H to aid the current discussion.

The first step in the code is to allocate arrays on the GPU's global memory to store the solution, potential, and temporary arrays. This is accomplished by using the `cudaMalloc()` or the `cudaMallocPitch()` (see Sec. 7.2.3) function. Nine arrays, the size of the total number of grid points $\mathbf{N}$, are allocated when using the RK4+CD scheme, while eleven are needed for the RK4+2SHOC scheme. The arrays are composed of the real and imaginary parts of the solution vector Ψ, one array for the (real) external potential $V(\mathbf{r})$, and the real and imaginary parts of the three required temporary arrays for the RK4 algorithm (see below). For the RK4+2SHOC scheme, two additional arrays are needed to store the real and imaginary parts of the second-order Laplacian D. As mentioned in Sec. 6.3, it has been shown that splitting the real and imaginary parts of the solution arrays directly is more efficient than using a complex number structure object [78].

The next step in the code is to transfer the solution Ψ and potential $V(\mathbf{r})$ arrays from the CPU's memory to the memory locations allocated for them on the GPU's global memory. This is done through the `cudaMemcpy()` or the `cudaMemcpy2D()` (see Sec. 7.2.3) function.

Before calling the compute kernel routines to integrate the NLSE, it is first necessary to define the CUDA compute-grid and block sizes for use in the kernel calls. The choice of block size is not trivial, and is dependent on the size of the problem and the GPU card being used. In order for a GPU to be most efficient, it requires that the problem have good *occupancy* on the specific card. This means that as many SMs are running as many SPs during the program simultaneously. Occupancy can have a large impact on performance. For example, if the number of threads in a block is smaller than the number

of SPs per SM, not all SPs on the SMs will be utilized. On the other hand, if the block size is too big, the problem might fit into very few blocks, in which case the other SMs on the GPU will not be used at all! In NLSEmagic, the block sizes are chosen to provide efficiency for typical sized problems in each dimension (see following sections for details). The block size in each dimension is set using the `dimBlock()` function, whose result is stored in an intrinsic variable `dimBlock` which is a structure of type `dim3`.

The compute-grid size is found by taking the `ceil` of the length of the dimension of the specified problem array divided by the block size of that dimension. For example, the x-direction grid size in two dimensions would be found by `ceil(M/dimBlock.x)` where `dimBlock.x` is the x-direction block-size (it is important to note that, as per the discussion in Sec. 6.3, the x- and y-direction of the CUDA blocks are actually along the y- and x-direction respectively of the solution Ψ within the C code). Once the values for the grid dimensions are found, the grid is set up using the function `dimGrid()`, whose output is stored in an intrinsic variable called `dimGrid` which is a structure of type `dim3`.

Once the CUDA compute-grid and block sizes are set up, the code computes a chunk-size number of time-steps of the RK4 scheme using kernels which run on the GPU. The more computations a single kernel call performs, the better the performance of the GPU code. Therefore, we try to combine as many steps in the RK4 algorithm of Eq. (3.1) as possible. Since each compute block must be able to run independently of all the other blocks within a single kernel call, synchronization and race conditions (i.e, making sure a required result is computed before used) will limit how many steps of the RK4 can be combined. For example, each computation of $F(\Psi)$ in Eq. (3.1) (after the first one), for each grid point, requires the previously completed computation of Ψ_{tmp} of the neighboring points from the previous step in order to compute the Laplacian of Ψ. Therefore, even with block-wide synchronizations, grid points along the boundary of compute-blocks will have no guarantee that the Ψ_{tmp} values it needs will have been computed yet or not (or possibly, even overwritten). Therefore, multiple separate kernels are required to run the

steps in Eq. (3.1). When using the RK4+2SHOC scheme, an additional kernel (called four times) is required due to the same synchronization issues involving neighboring points needed from one step to the other (in this case the values of the Laplacian D in the first step of the 2SHOC scheme).

Although the computations of $F(\Psi)$ and D require separate kernels, the other steps in the RK4 algorithm *can* be combined with the kernels for $F(\Psi)$. Thus, the RK4 can be computed using only four kernel calls (8 when the additional kernel needed in the 2SHOC scheme is included) which contain steps 1–2, 3–5, 6–8, and 9–10 of Eq. (3.1). The only problem is that steps 3–5 and 6–8 can cause a synchronization issue. This is because step 5 (and 8) overwrites the Ψ_{tmp} values, which are the inputs to step 3 (and 6). Thus, when one block of threads finishes the step 3–5 computation, the adjacent block could still require the old values of Ψ_{tmp} along its neighboring edge to compute the Laplacian. To solve this, we introduce a storage array called Ψ_{out} which allows step 5 to output its values to Ψ_{out} so that the other blocks still have access to the original values of Ψ_{tmp} . Then in step 6, the code evaluates $F(\Psi_{\text{out}})$, while step 8 outputs to Ψ_{tmp}. Adding this vector does not increase the overall global memory storage requirements of the RK4 scheme because since the steps are now combined into four kernels, the K_{tmp} array no longer needs to be stored in global memory and can instead be stored in a per-block shared memory. The RK4 algorithm can therefore be described on the GPU as

$$
\begin{aligned}
&1)\ k_{\text{tmp}} = F(\Psi) && 7)\ k_{\text{tmp}} = F(\Psi_{\mathbf{out}}) \\
&2)\ K_{\text{tot}} = k_{\text{tmp}} && 8)\ K_{\text{tot}} = K_{\text{tot}} + 2\,k_{\text{tmp}} \\
&3)\ \Psi_{\text{tmp}} = \Psi + \frac{\mathbf{k}}{2}\,k_{\text{tmp}} && 9)\ \Psi_{\text{tmp}} = \Psi + \mathbf{k}\,k_{\text{tmp}} \\
&4)\ k_{\text{tmp}} = F(\Psi_{\text{tmp}}) && 10)\ k_{\text{tmp}} = F(\Psi_{\text{tmp}}) \\
&5)\ K_{\text{tot}} = K_{\text{tot}} + 2\,k_{\text{tmp}} && 11)\ \Psi = \Psi + \frac{\mathbf{k}}{6}\,(K_{\text{tot}} + k_{\text{tmp}}), \\
&6)\ \Psi_{\mathbf{out}} = \Psi + \frac{\mathbf{k}}{2}\,k_{\text{tmp}}, &&
\end{aligned}
\tag{7.1}
$$

where k_{tmp} is only stored in shared memory, while Ψ, K_{tot}, Ψ_{tmp}, and Ψ_{out} are stored in global memory (although during computation, they are transfered to shared memory as discussed below).

All four kernel calls in the computation of a RK4 step are computed using a single kernel function named `compute_F()`. One of its input parameters tells the kernel which step it is computing, and the corresponding computation is selected in a switch statement. For the 2SHOC scheme, an additional kernel is written named `compute_D()` which computes the D array before each call to `compute_F()`.

When a kernel is launched, the blocks in the compute-grid are scheduled to run on the MP, where each thread in the block runs the kernel. The first step in any kernel is to calculate the running thread's index position based on the intrinsic variables described in Sec. 7.1.2. For example, in one-dimension, the thread's position in the block is given by `threadIdx.x` while its position in the grid is given by `blockIdx.x*blockDim.x + threadIdx.x`.

As discussed in Sec. 7.1.1, thread accesses to the per-block shared memory are much faster than accesses to the global memory of the GPU. Therefore if any array needed in the computations is accessed more than once by a thread in the block, it is worthwhile to copy the block's required values of the array from global memory into shared memory. Then, after computing using the shared memory, the results can be copied back into global memory. Since shared memory is block-based, each thread in the block needs to copy its own value from the global array into the shared memory space (some threads may need to copy more than one value as we will show in Secs. 7.2.2, 7.2.3, and 7.2.4).

In `compute_F()`, five shared memory arrays are required (seven for the 2SHOC scheme). These consist of the real and imaginary parts of the Ψ input and the $F(\Psi)$ result [called k_{tmp} in the RK4 algorithm of Eq. (7.1)], as well as the potential array V. In the 2SHOC scheme, the real and imaginary parts of D are also stored in shared memory

arrays. In the `compute_D()` kernel for the 2SHOC scheme, only two shared memory arrays are needed (for Ψ).

Each thread in the block copies the global memory values into the shared memory arrays. Since the number of stream processors in the stream multi-processor that the block is being computed on almost always has less processors than the number of threads in the block, threads will typically not have access to all of the shared memory of the block after copying their own values. This is a problem because each thread has to access neighbor elements in the shared memory block for computation of the Laplacian of Ψ. Therefore, a `__syncthreads()` function is called after the copy which synchronizes all the threads in the block, ensuring that the entire shared-memory array is filled before using it.

After the synchronization, the threads on the boundary of the block have to copy additional values into shared memory since they require accessing points which are beyond the block boundary due to the finite-difference stencil (see Secs. 7.2.2, 7.2.3, and 7.2.4 for details in each dimension). These transfers are not done before the block-synchronization because in CUDA, when a group of threads all need to perform the same memory transfer, they are able to do it in a single memory copy instead of one-by-one. Adding the boundary transfers before the synchronization may break up this pattern and cause the memory-copies not to be aligned for single-instruction copying.

After the transfers to shared memory are completed, the $F(\Psi)$ values are computed for all grid points within the boundaries of the solution Ψ. Once these interior points of $F(\Psi)$ are computed, threads which happen to be on the boundary of Ψ compute the boundary conditions of $F(\Psi)$. When using the MSD boundary condition of Chapter 4, a block-synchronization is required to be sure that the interior value of $F(\Psi)$ has been pre-computed (since it is necessary for the computation of the MSD condition). A problem exists if in any direction, the block is only one cell wide (since the interior point of $F(\Psi)$ will not be computed by that block). To avoid this problem in the most efficient way (i.e, without adding extra error-checking code), we leave the detection and solution of

this issue to the NLSEmagic driver scripts which automatically adjust the solution grid size to ensure that this condition does not occur.

Once the boundary conditions of the $F(\Psi)$ array is completed, the kernel uses the result to compute the remaining sub-steps of the RK4 algorithm. As mentioned above, which step in the RK4 is being computed is known through an input parameter to `compute_F()`.

For the 2SHOC scheme, the `compute_D()` kernel is very similar in structure to `compute_F()`. It first copies values of Ψ from global to shared memory (here only two shared memory arrays are required) including block-boundary values, and then computes D with its boundary conditions and stores the result back into global memory.

After the desired number of RK4 time-steps (the chunk-size) have been completed, a call to `cudaDeviceSynchronize()` is made which ensures that all kernels are fully completed before continuing in the host code (before the CUDA SDK 4.0, this function was called `cudaThreadSynchronize()`). This is required because although the CUDA GPU-to-CPU memory copies are designed to be implicitly synchronizing, in practice for large problems, we found that this was not completely reliable and therefore we explicitly synchronize the kernels. The current Ψ arrays are then transfered to the CPU using the `cudaMemcpy()` (or `cudaMemcpy2D()`) function. Finally, all the CUDA global memory spaces used are freed with `cudaFree()` and the MEX file returns the new value of Ψ back to MATLAB.

7.2.2 One-Dimensional Specific Code Design

In the one-dimensional NLSEmagic integrators, the CUDA compute-grid is set-up to be one-dimensional with a block size of 512 threads. Although for small problems, a block size of 512 is not efficient in terms of occupancy (see Sec. 7.2), we feel that since one-dimensional problems typically run fast enough without any GPU-acceleration, the only time GPU-acceleration will be needed in one-dimensional problems is when the

problem size is very large. In such a case, a block size of 512 will allow for good occupancy in most situations.

One of the key concerns in parallelizing finite-difference codes is the need for the cells on the edge of a compute-block to require values from their neighbor in another block. Since in the CUDA codes all threads in all blocks have equal access to global memory, there seems to be no problem with threads on the block boundaries. However, this is only true if the CUDA code only uses global memory (see Ref. [77] and [78], where the authors take this approach).

In order to greatly speedup the codes, it is vital to use the per-block shared memory in the computation whenever any global memory space is needed to be accessed by any thread more than once. The typical way to use shared memory is to have each thread in the block copy one value of Ψ (or D) and V from global memory into shared memory arrays which are the same size as the block size. Then the computation is done using the shared memory array, and at the end each thread stores the output back into the global memory array. However, in this scenario, the block boundary cells do not have all their required Ψ (or D) values available in shared memory. There are a few different ways to deal with this. A simple solution is to have the block boundary threads directly access the required neighbor point(s) of Ψ (or D) from global memory when needed (as was done in Ref. [75]). Another solution is to set up the grid to overlap the boundary cells so that all threads copy values into shared memory as before, but only the interior threads of the block perform the computations, which can be performed using only shared memory. Through testing, we have found that this solution is not efficient because a large number of threads (the boundary threads) become idle after the memory copy.

Another solution is that instead of having the shared memory size be equal to the block size, the shared-memory size is allocated to be two cells greater than the block size. In the stage of the kernel where the threads transfer Ψ and V from global memory into shared memory, the boundary threads also transfer the neighboring points into the extra

cell space at the boundary of shared memory. Therefore there are two global memory transfers performed by the block boundary cells instead of one. This process is depicted in Fig. 7.3. Later, in the computation part of the kernel, all threads can then compute using

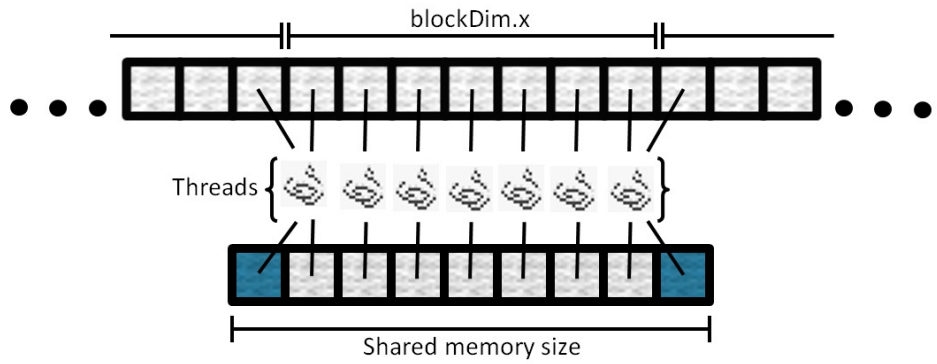


Figure 7.3. One-dimensional shared memory structure and transfer of global memory to shared memory. Note that the threads on the boundary of the block must perform two global memory retrievals into shared memory.

shared memory. Although this seems to be equivalent to the first method in terms of the number of global memory accesses (since in the first solution, the boundary cells access global memory twice as well — once to transfer their value to shared memory and once to access their required neighboring point), in practice, due to intricacies of the card hardware and software execution (for example, limiting the number of divergent branches), this method can be more efficient overall and therefore the one adopted in our codes.

7.2.3 Two-Dimensional Specific Code Design

In two-dimensional CUDA codes, the CUDA API provides special global memory allocation and transfer routines called `cudaMallocPitch()` and `cudaMemcpy2D()`. The `cudaMallocPitch()` allocates a global memory space which is expected to be accessed in a two-dimensional pattern. It aligns the data with the card hardware, often padding the rows of the matrix. Therefore, the memory allocation function returns a pitch value which is used to access the element (i, j) as `pitch*i+j` instead of the usual `M*i+j`. The pitch

value is also used in the `cudaMemcpy2D()` to transfer data from the CPU to the GPU and vice-versa. The special two-dimensional routines are somewhat optional as one could also just use a standard linear memory transfer of the data and access the elements normally, but NVIDIA strongly recommends against it [109]. Therefore, the two-dimensional integrators of NLSEmagic use the specialized functions.

The CUDA compute-grid is set to be two-dimensional with two-dimensional blocks for easy indexing. The block size is set to be 16×16. Based on performance tests, this is the best overall block size to use even though Fermi cards would allow a larger block size. The occupancy for a 16×16 block size is quite good even for smaller problems.

As in the one-dimensional code, the shared memory space is allocated to be two cells larger in width and height of the block size as shown in Fig. 7.4. The boundary

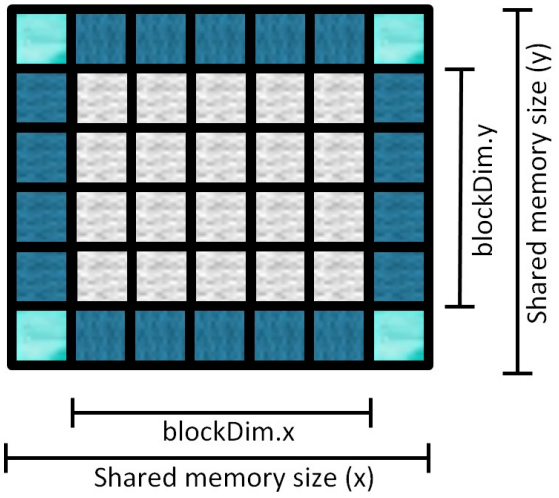


Figure 7.4. Two-dimensional shared memory structure. The threads on the boundary of the block must perform additional global memory retrievals into shared memory. Those on the corners perform a total of three transfers for the CD scheme, and four transfers for the 2SHOC (due to the required corner cells in the 2SHOC scheme). All other boundary threads perform two total transfers.

threads once again grab the neighboring points in addition to their designated points from global memory during the shared-memory transfer process. For the CD scheme, the

corner neighbor points are not needed, but for the 2SHOC scheme they are. This adds an additional global memory access on the corner threads (in addition the extra accesses from the other two neighboring points), making the corner threads copy up to four global values per array into shared memory.

When dividing the problem into CUDA blocks, the CUDA thread access ordering is an important consideration. The CUDA grid and block indexing is column-major ordered, with the 'x' direction along the columns and the 'y' direction is along the rows, and the threads are scheduled in that order. Thus, when copying arrays from global memory to shared memory, it is important that the access pattern of the memory being copied matches the access pattern of the thread scheduler [125]. Since C arrays are row-major, a shared memory array `A` should be allocated (counter-intuitively) as `A[BLOCK_SIZEY][BLOCK_SIZEX]`. The memory copy from elements in a global array `A_global` is then given as

```
A[threadIdx.y][threadIdx.x] = A_global[pitch*i+j],
```

where the position of the current thread within the solution array is given by

```
j = blockIdx.x*blockDim.x + threadIdx.x,
i = blockIdx.y*blockDim.y + threadIdx.y.
```

Since in MATLAB (when using `meshgrid()` to form the x and y arrays of Ψ) the x direction of Ψ is along the rows of the Ψ array, and the y direction is along the columns, and when in the MEX file, this is transposed, then, as mentioned in Sec. 6.3, the 'x' direction of the CUDA grid and blocks actually represent the y direction in Ψ and vice-versa. Therefore, in defining the grid size, we use `M/dimBlock.x` for the x dimension of the grid and `N/dimBlock.y` for the y even though `N` is the x-direction of Ψ and `M` is the y-direction. This is very important to remember when checking the MSD boundary condition grid requirement mentioned in Sec. 7.2.1. The thread access ordering described is not a trivial matter. In our tests, using the incorrect ordering can slow down the codes by almost a half!

7.2.4 Three-Dimensional Specific Code Design

In three dimensions, the CUDA API provides a hardware-optimized memory allocation and transfer functions similar to the two-dimensional ones mentioned in Sec. 7.2.3. However, the implementation of the three-dimensional functions (called `cudaMalloc3D()` and `cudaMemcpy3D()`) are not straight-forward and requires making structures with information about the arrays, and passing the structures to the functions. Through testing of the two-dimensional NLSEmagic codes versus a naive two-dimensional implementation (without the specialized allocation and memory-copy), despite the recommendations of NVIDIA, we did not observe any significant performance boost in using the hardware-optimized routines. Therefore, in order to keep things as simple as possible, the three-dimensional integrators of NLSEmagic do not use the specialized routines, and instead use the standard `cudaMalloc()` and `cudaMemcpy()` used in the one-dimensional codes in which case a global memory array `A_global` for a point (i, j, k) is simply given by `A_global[N*M*i + M*j + k]`.

In Refs. [76] and [77], the three-dimensional finite-difference CUDA codes were set-up to do multiple two-dimensional sweeps and combine the results to form the three-dimensional derivatives. Thus, only two-dimensional grid and block structures were used. This was done because the limit of 16KB of shared memory of the older GPUs was felt to be too limiting for using three-dimensional blocks. However, in our NLSE integrators, we use three-dimensional blocks which we note is efficient even when using older Tesla GPUs. On the Fermi cards, we have tested our code for various block sizes and have determined what are the most efficient block sizes for a typical problem size depending on the numerical method and precision being used (sometimes the choices are based on the shared memory size limitation of 48KB). For single precision, we use a block size of $16 \times 16 \times 4$ for the RK4+CD scheme and $12 \times 12 \times 4$ for the RK4+2SHOC. In double precision we use a block size of $10 \times 10 \times 5$ for the RK4+CD scheme and $8 \times 8 \times 6$ for the RK4+2SHOC.

An inherent difficulty with three-dimensional CUDA codes is that originally, even though blocks could be three-dimensional, the CUDA compute-grid could only be two-dimensional. Thus, one could not implement three-dimensional codes in a straight-forward extension from two-dimensional grid structures (i.e, to have a three-dimensional grid with each thread having an intrinsic `blockIdx.z` variable). This is an additional reason why previous studies (such as Refs. [76] and [77]) used two-dimensional slice algorithms.

Recently, NVIDIA has solved this problem as of the release of the CUDA SDK 4.0. With the new SDK, all Fermi based GPUs can now have a full three-dimensional grid structure. Our NLSE integrators in NLSEmagic were developed before this release, and therefore did not take advantage of this new feature. Instead, a custom logical three-dimensional structure on a two-dimensional CUDA compute-grid was used. In order to allow NLSEmagic to be compatible with older Tesla-based GPUs, the integrators are not updated to use the new three-dimensional grids.

The three-dimensional problem grid is viewed as a series of three-dimensional slabs, each being one block high. These slabs are then seen as being placed side-by-side to create a two-dimensional grid of three-dimensional blocks as shown in Fig. 7.5. We thus

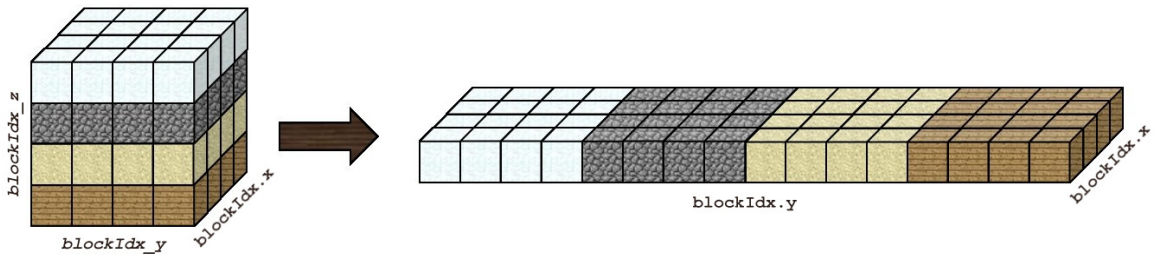


Figure 7.5. Logical block reordering for a 2D CUDA grid from desired 3D grid structure.

have a CUDA compute-grid which is two-dimensional with three-dimensional blocks, and a ‘virtual’ compute-grid which is a three-dimensional grid with three-dimensional blocks. In order to make the indexing as easy as possible within the actual CUDA compute-grid,

we create new indexing variables per thread in order to use the full three-dimensional virtual compute-grid.

We note here that, as in the two-dimensional case, the CUDA block/thread directions x, y, and z do not all correspond to those of Ψ. The CUDA block x-direction is actually along the y-direction of Ψ (M) and the CUDA block y-direction is actually along the x-direction of Ψ (N). The z-direction of both are equivalent. This is important to keep in mind for the computation of grid sizes and in checking the block requirements of the MSD boundary-condition.

The first step in setting up the three-dimensional grid is that a variable

```
gridDim_y = ceil(N/dimBlock.y)
```

is set to represent the virtual y-direction grid size. Therefore, the actual CUDA grid is defined to be

```
dimGrid(ceil(M/dimBlock.x),gridDim_y*ceil(L/dimBlock.z)).
```

The value of `gridDim_y` is passed into the kernels in order for the threads to have access to it. Inside the kernel, a new variable representing the thread's z-direction block position within the virtual three-dimensional grid called `blockIdx_z` is created defined as

```
blockIdx_z = blockIdx.y/gridDim_y,
```

which uses integer division which acts as a `floor` function. The y-direction block position in the virtual grid is then found and stored in a new variable as

```
blockIdx_y = blockIdx.y - blockIdx_z*gridDim_y.
```

The `blockIdx_y` variable is therefore replacing the intrinsic `blockIdx.y` variable (which would give the y-direction block within the true CUDA grid). These new block position variables are depicted in Fig. 7.5.

Once the new indexing variables are created, the computation of the thread's designated position within the solution array is straight-forward as if there were a true CUDA three-dimensional grid. The values of the thread's indexes is therefore

```
k = blockIdx.x*blockDim.x + threadIdx.x,
j = blockIdx.y*blockDim.y + threadIdx.y,
i = blockIdx.z*blockDim.z + threadIdx.z,
```

where the thread is associated with the global memory position `A_global[N*M*i + M*j + k]`.

Once the older Tesla GPU cards are no longer in general use, direct three-dimensional CUDA grids can be implemented into the code, which, for our implementation, would be very straight-forward (just setting up the grid based on M, N, and L directly and replacing the new thread indexes with the intrinsic ones).

As was the case in the two-dimensional codes, the dimensions of the shared memory space must be done in accordance with the thread access patterns in order to be efficient. (Note that this discussion is completely independent of the implementation of the three-dimensional compute-grid since, either way, three-dimensional blocks are being used and this issue is a block-wide problem). Since the threads are accessed in column order, first along x, then y, then z, it is important that the shared memory space be allocated as `A[BLOCK_SIZEZ][BLOCK_SIZEY][BLOCK_SIZEX]` so that the memory copy is done as

```
A[threadIdx.z][threadIdx.y][threadIdx.x] = A_global[N*M*i + M*j + k].
```

As in the two-dimensional case, this correct ordering has a huge effect on performance (upwards of 50%) and is therefore vital.

As in the lower-dimensional integrator codes, the three-dimensional integrators allocate the shared memory space to be two cells larger in each dimension than the block size. Thus, the threads on the block boundaries copy additional values from global memory into shared memory. For each required shared memory array. the interior threads copy one value, the threads along the faces of the block boundaries copy two values, the threads along the edges of the block boundary copy three values (four for Ψ in the 2SHOC scheme), and the threads on the corners of the block boundary have to copy four values

(seven for Ψ in the 2SHOC scheme). Since the 2SHOC scheme never relies on the diagonal corner cells of Ψ or D, as it does not use a full 27-points stencil as seen in Eq. (3.27), no additional copies are necessary. Due to the large number of extra serialized memory copies for the boundary cell threads, the performance of the three-dimensional codes is expected to be less than the two-dimensional codes (this is indeed the case as shown in Sec. 7.3.4).

7.3 SPEEDUP RESULTS

Here we show the results of comparing the compute-time of the NLSEmagic CUDA integrators versus the serial MEX integrators. We choose to run the codes on a GeForce GTX 580 card (full specifications, as well as specifications of the CPU used are listed in Appendix G). Even though the double precision performance is artificially reduced in the GeForce cards, we show that this only effects our one-dimensional code, whereas our two- and three-dimensional codes have memory bottlenecks which make the reduced double precision performance negligible in practice. (After possible future optimization of the code, this may not be the case and it would be beneficial to run the codes on a more expensive Tesla compute-only GPU). Because the GeForce GPUs are many times cheaper than the Tesla (or Quadro) cards and the performance is not expected to be effected in a major way, we feel justified in running our codes on a GeForce card.

For all speedup tests, we use the example simulations of Sec. 6.1 with an end-time of $t = 50$ and vary the total grid-size from about 1000 to $3,000,000$. In each case we compute the computation time of the integrators, ignoring the small extra time required for generating the initial condition and outputting results.

Before computing the speedup results, it is first necessary to examine the effect that CPU-GPU memory transfers has on the performance of the integrators.

7.3.1 Chunk-size Limitations

As mentioned in Sec. 7.1.1, memory transfers between the CPU and GPU are very slow, and therefore it is best to compute as much as possible on the GPU before transferring the data back. For simulations where analysis results are cumulative, the entire simulation can be performed on the GPU with only two memory transfers (one at the beginning and one at the end of the computation). For most studies of time-dependent problems such as the NLSE, it is desired (or essential) to have access to the computation data at regular intervals in order to save the data, display it for observation and animations, and to run intermediate analysis. However, the more times the data is needed by the CPU, the slower the code will perform. Therefore for NLSEmagic, the more frames the simulation is plotted and analyzed, the slower the overall code will be.

In order to use the codes in the most efficient manner, it is necessary to see how much the simulation is slowed down as a function of the size of the chunk of time-steps performed by the CUDA integrators (which we have designate as the *chunk-size*). Once a chunk-size is found which exhibits acceptable lack of slow-down, the number of frames that the solution is viewable will be determined by the number of time-steps. Thus, if one wants more frames for a particular simulation, one could lower the time-step increasing the number of time-steps taken, or accept the reduced performance. Which of these two options is more efficient would have to be determined.

To see how the chunk-size affects performance of the NLSEmagic CUDA integrators, we run the examples described in Sec. 6.1 for various chunk-sizes and compare the compute-times. Since the performance of the codes due to chunk-size is not affected by the total number of time steps (as long as the number of time steps is larger than several chunk-sizes), we fix the end-time of the simulations to be $t = 5$ and the time-step to be $\mathbf{k} = 0.005$ yielding $1,000$ time-steps. We run the examples in each dimension with single and double precision for both the RK4+CD and RK4+2SHOC schemes for total grid sizes ($\mathbf{N}$) around $1,000$, $10,000$, $100,000$, and $1,000,000$ (the two-

and three-dimensional codes have slightly different resolutions due to taking the floor of the square- or cubed-root of N as the length in each dimension respectively). To eliminate dependency of the results on the specific compute times of the given problem (which would make comparison of different grid sizes difficult), for each chunk-size, we compare the compute times of the simulation to the fastest time of all the simulations of that problem. As expected, this fastest time nearly always occurs when the chunk-size is equal to the number of time-steps. We thus compute a 'slowdown' factor as the time of the fastest run divided by the time of the other runs. This makes the results applicable to almost all simulations run with NLSEmagic. The results are shown in Fig. 7.6. We see that in each case, the chunk-size has to be somewhat large for the codes to perform with top efficiency. Simulations with lower chunk-sizes can be done with acceptable slowdown based on the figures. In the higher dimensional cases, the larger the total grid size, the less the slowdown factor for a given chunk-size (in the one-dimensional case, the results are varied, with the lowest resolution tested outperforming the highest). Also, in general, it is seen that the higher-dimensional codes have less slowdown for a given chunk-size than lower-dimensional codes. Likewise, using the 2SHOC versus the CD scheme and double versus single precision, lower the slowdown for a given chunk-size as well. This is understandable since the more work the CUDA kernels are doing, the higher percentage of the total time is spent on computations compared to memory transfers to the CPU and back. In either case, when running NLSEmagic, it is best to first look up on Fig. 7.6 where the problem being simulated falls, and then choose an appropriate chunk-size to get the best performance out of the codes (within the simulation requirements).

7.3.2 One-Dimensional Speedup Results

For the one-dimensional results, we use the dark soliton solution in Sec. 6.1 with an end-time of $t = 50$. The solution is plotted five times during the simulation yielding a chunk-size of $5,000$, which is well over the efficiency requirements as discussed in Sec. 7.3.1. The soliton is simulated with a grid size which varies from $N = 1,000$ to

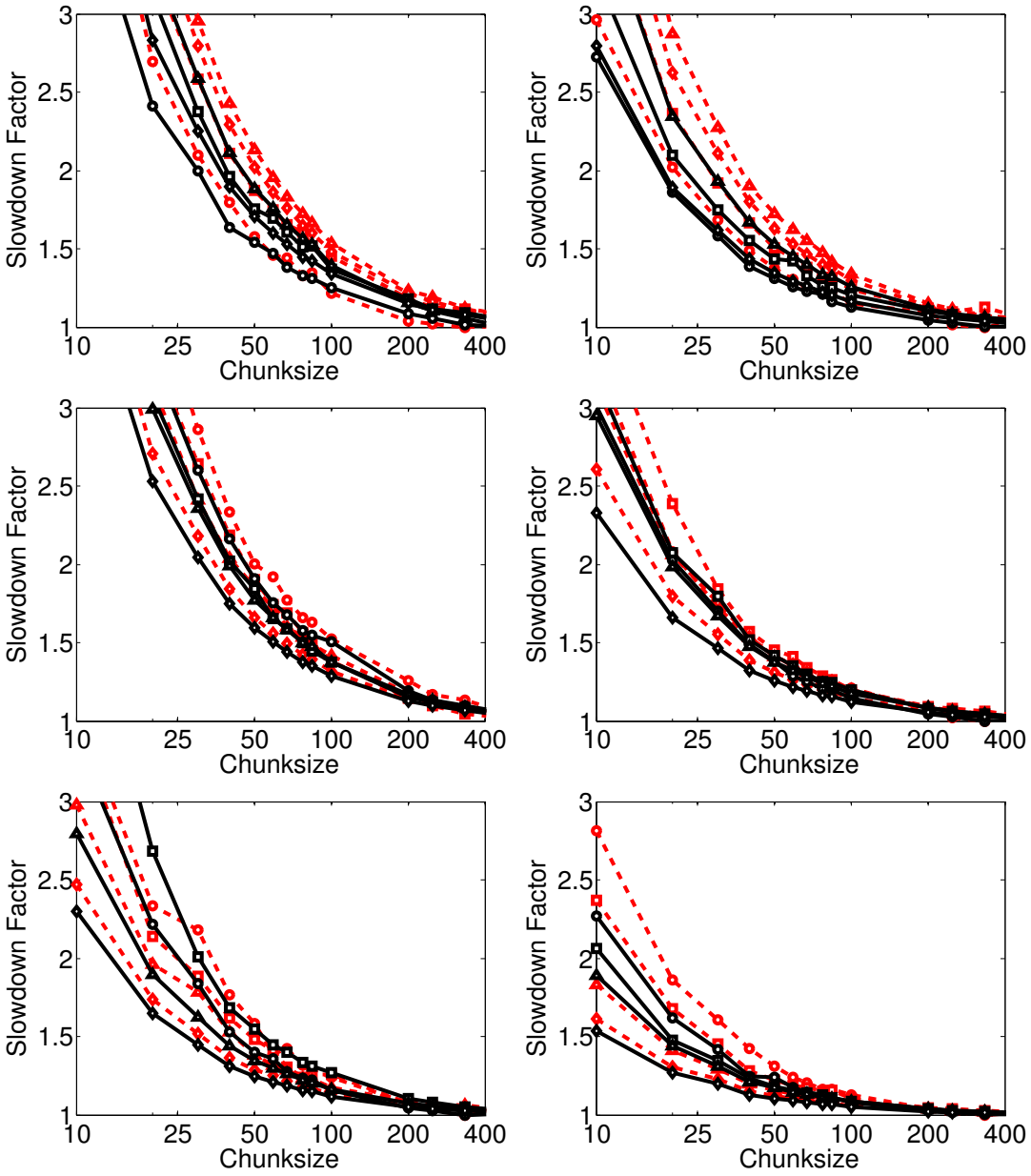


Figure 7.6. Slowdown factor as a function of chunk-size of the CUDA NLSEmagic integrators computed using the examples of Sec. 6.1 with an end-time of $t = 5$. The slowdown is defined as how much slower the simulation is compared to the fastest possible simulation (where the chunk-size equals the number of time-steps) not shown. The results are shown for the RK4+CD and RK4+2SHOC (left and right respectively), for one-, two-, and three-dimensional simulations (top to bottom respectively), and for single and double precision (dashed and solid lines respectively). The circle, square, triangle, and diamond lines represent total grid-sizes N of 1000, $10,000$, $100,000$, and $1,000,000$ respectively.

$N = 3,000,000$. Validation of the codes is done by comparing the solution to the known exact solution. The simulation compute-times and speedups compared to the serial MEX integrators are shown in Fig. 7.7 and Table 7.1. The best results are those of using the CD+RK4 scheme in single precision where we observed speedups around 90 for large grid sizes. Since, as was shown in Fig. 6.4, the serial integrators are about 12 times faster than the MATLAB script codes for those resolutions, the NLSEmagic CUDA MEX codes for the one-dimensional CD+RK4 scheme in single precision for large resolutions are about *1000 times faster* than the equivalent MATLAB script code. In terms of actual time for the simulation tested, this would equate to taking roughly 30 seconds using the GPU code, 40 minutes using the serial MEX code, and 8 *hours* 20 minutes using the MATLAB script code.

Since the block size for the integrators was chosen to be 512, it is understandable that the compute time for the CUDA MEX codes stays quite low until resolutions of $10,000$ or so. This is because the GPU being used has 16 MPs, and therefore resolutions up to 8000 can be computed in one sweep of the GPU MPs, while higher resolutions require multiple blocks to be computed on the same MP.

It is noticeable that the double-precision performance is almost half that of the single precision. This is partly due to what was mentioned in Sec. 7.1.1 that the GeForce cards have their double precision FLOP count artificially reduced by three-quarters, making the FLOP count one-eighth that of the single precision performance. In addition, memory transfer is considered to be a large factor in code performance, and since double-precision variables take twice the memory space as single precision, a reduction in performance is understandable.

It is also apparent that the speedup when using the 2SHOC scheme is lower than the CD scheme. This is also understandable because in the 2SHOC scheme, kernels which only compute the D array (the standard second-order Laplacian) are needed, and they have a smaller amount of floating-point operations than the kernels which compute the full

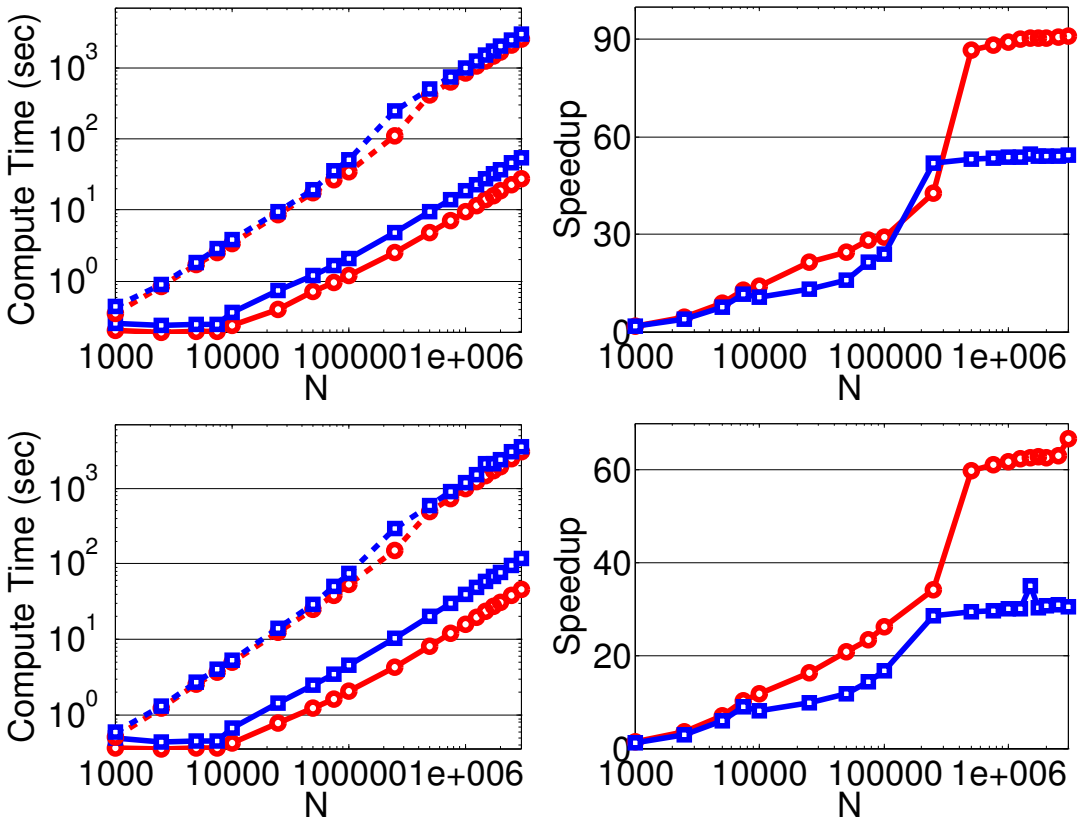


Figure 7.7. Computation times (left) and speedups (right) of the one-dimensional NLSEmagic CUDA MEX integrators (solid) versus the serial MEX integrators (dashed) for simulating the dark soliton solution described in Sec. 6.1 with an end-time of $t = 50$. The results are shown for both the RK4+CD scheme (top) and the RK4+2SHOC scheme (bottom) using both single (red circles) and double (blue squares) precision.

Table 7.1. Subset of the one-dimensional timing results from Fig. 7.7. The results are shown for the RK4+CD scheme in single (s) precision (best results) and for the RK4+2SHOC in double (d) precision (weakest results).

	GPU Time (sec)		CPU Time (sec)		SPEEDUP	
N	CD (s)	2SHOC (d)	CD (s)	2SHOC (d)	CD (s)	2SHOC (d)
1000	0.21	0.49	0.35	0.61	1.69	1.23
10000	0.24	0.67	3.39	5.38	14.07	7.99
100000	1.21	4.52	35.15	75.31	29.12	16.65
1000000	9.41	39.72	838.21	1192.27	89.08	30.01
3000000	27.65	117.32	2515.56	3581.40	90.98	30.53

NLSE and RK4 step. Since the amount of computations in the D kernel is so small but the amount of required memory transfers is comparable to those of $F(\Psi)$, the speedup is smaller. As will be shown in Secs. 7.3.3 and 7.3.4, in higher dimensions, this issue is somewhat minimized due to the added number of points in the second-order Laplacian stencil (the speedup reduction is lowered from 27% in one dimension to 16% in three dimensions in single precision, and from 45% to 0% in double precision).

A useful way of representing the timing results from Fig. 7.7 is to show the number of RK4 steps per second that the simulations took. The reason this can be useful is that typically, in a real problem using the RK4 scheme, one first chooses the domain size needed and the grid resolution desired (taking into account the spatial accuracy of the scheme, and the number of grid points needed to adequately resolve the solution). Once the spatial grid-spacing is chosen, one computes the maximum possible stable time-step. Now, based on the simulation time desired, the number of time-steps the simulation will take can be found. Therefore, knowing how many RK4 steps per second the codes are capable of running on the hardware will give a good estimate of the total time needed for the simulation. We show the number of RK4 steps per second from the simulations done for Fig. 7.7 in Fig. 7.8. An added benefit is that the results shown in Fig. 7.8 are

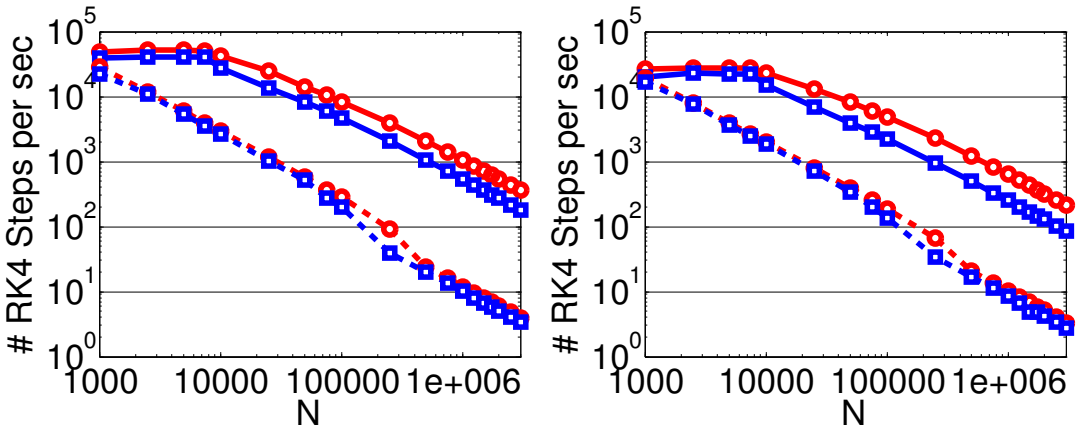


Figure 7.8. The number of RK4 time-steps per second for the simulations of Fig. 7.7 for the RK4+CD (left) and RK4+2SHOC (right) schemes. The additional figure properties are the same as those in Fig. 7.7.

independent of the total number of time-steps chosen for the simulations (keeping the chunk-size issues addressed in Sec. 7.3.1 in mind).

7.3.3 Two-Dimensional Speedup Results

For the two-dimensional tests, we use the unoptimized steady-state dark vortex approximate solution of Sec. 6.1. Since there is no analytical solution to the dark vortex, and moreover, since we are using only an approximation of the true solution, we cannot record the error of the runs. Therefore, in order to validate the simulations, we displayed near-center cell data points for each frame and compared them with those from the output of the non-CUDA MEX codes, and checked that they were equivalent.

Like the one-dimensional tests, the solution is plotted five times during the simulation yielding a chunk-size of 5000, which is well over the efficiency requirements discussed in Sec. 7.3.1. The vortex solution is simulated with a total grid size which varies from about $\mathbf{N} = N * M \approx 1000$ to $\mathbf{N} \approx 3,000,000$. The N and M dimensions are determined by taking the floor of the square-root of $\mathbf{N}$ and are sometimes slightly altered due to the block requirements of using the MSD boundary condition (see Sec. 7.2.1).

The simulation compute-times and speedups compared to the serial MEX integrators are shown in Fig. 7.9 and Table 7.2. There is noticeably less speedup in the two-dimensional codes when compared to the speedup of the one-dimensional codes. A possible explanation for the cause of the reduced speedup is that in two dimensions there are many more boundary cells versus interior cells of the CUDA blocks (in one dimension there are $O(1)$ while in a two-dimensional square block, there are $O(N)$). Since, as described in detail in Sec. 7.2.3, block cell boundaries require additional global memory accesses (especially the corners), a decrease in performance was excepted. Also, the number of total boundary grid points is much higher in two dimensions, in which case there are more threads computing boundary conditions than in one dimension. This can cause less speedup in the codes since the MSD boundary condition requires an extra block-synchronization not needed in the internal scheme. Slightly better speedup results

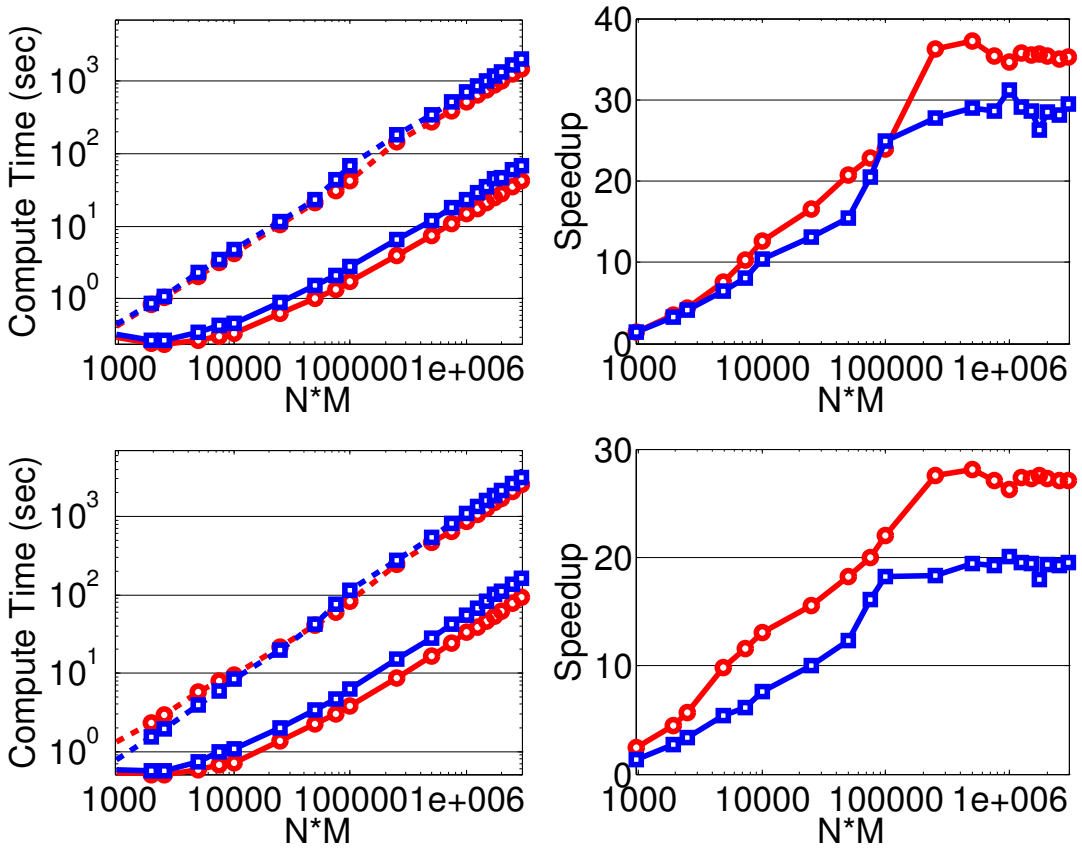


Figure 7.9. Computation times (left) and speedups (right) of the two-dimensional NLSEmagic CUDA MEX integrators (solid) versus the serial MEX integrators (dashed) for simulating the approximate dark vortex solution described in Sec. 6.1 with an end-time of $t = 50$. The results are shown for both the RK4+CD scheme (top) and the RK4+2SHOC scheme (bottom) using both single (red circles) and double (blue squares) precision.

Table 7.2. Subset of the two-dimensional timing results from Fig. 7.9. The results are shown for the RK4+CD scheme in single (s) precision (best results) and for the RK4+2SHOC in double (d) precision (weakest results).

	GPU Time (sec)		CPU Time (sec)		SPEEDUP	
$N * M$	CD (s)	2SHOC (d)	CD (s)	2SHOC (d)	CD (s)	2SHOC (d)
961	0.29	0.59	0.42	0.78	1.43	1.32
10000	0.34	1.09	4.25	8.35	12.61	7.65
99856	1.76	6.25	42.20	113.96	23.99	18.23
1000000	14.94	54.65	516.65	1094.72	34.59	20.03
2999824	42.15	162.79	1485.88	3173.51	35.26	19.49

could be obtained by choosing a problem that has Dirichlet boundary conditions, but it is important to show results that apply to a more general range of applications.

It is important to note that despite the reduction in speedup of the two-dimensional codes, the speedup observed is still quite high, especially considering the cost of the GPU card. In fact, in the RK4+CD test of $\mathrm{N} = 1732^2 = 2999824$, the GPU code took about 40 seconds, while the serial MEX code took over 24 minutes. Based on the MEX speedups of Sec. 6.3.1, the equivalent MATLAB script code would be expected to take almost 5 *hours* to complete the same simulation. It is also important to note that resolutions for large N values (where the best speedups are observed) are more common in two dimensions. Thus, the higher speedup results are expected to be more common in actual applications than those in one dimension.

As explained in Sec. 7.3.2, it is useful to see the timing results displayed as the number of RK4 steps per second. In Fig. 7.10 we show the RK4 steps per second of the results from Fig. 7.9.

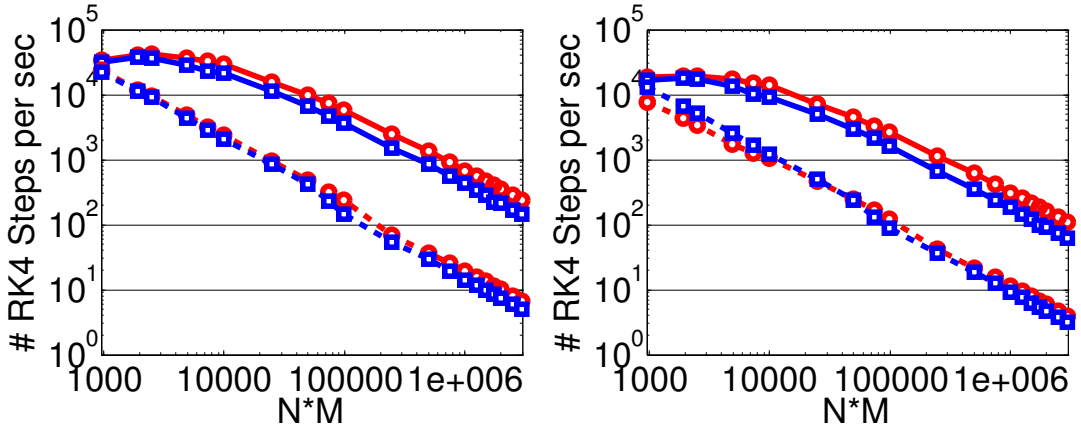


Figure 7.10. The number of RK4 time-steps per second for the simulations of Fig. 7.9 for the RK4+CD (left) and RK4+2SHOC (right) schemes. The additional figure properties are the same as those in Fig. 7.9.

7.3.4 Three-Dimensional Speedup Results

For the three-dimensional timings, we use the unoptimized approximate co-moving dark vortex ring of Sec. 6.1. As was the case in the two-dimensional tests, we cannot record the error of the simulations. Therefore, to validate the simulations, we once again displayed near-center cell data points for each frame and compared them with those from the output of the non-CUDA MEX codes.

Like the one- and two-dimensional tests, the solution is plotted five times during the simulation yielding a chunk-size of 5000, which is once again well over the efficiency requirements discussed in Sec. 7.3.1. The vortex ring is simulated with a total grid size which varies from $\mathbf{N} = N * M * L \approx 10,000$ to $\mathbf{N} \approx 3,000,000$. The N, M, and L dimensions are determined by taking the floor of the third-root of $\mathbf{N}$ and then slightly adjusted as needed to comply with the block requirements of using the MSD boundary condition mentioned in Sec. 7.2.1. Although for the one- and two-dimensional tests we started with simulations of size $\mathbf{N} \approx 1000$, in the three-dimensional case this was not possible due to the size of the vortex ring and the chosen spatial step-size (the vortex ring overlaps the grid boundaries when $\mathbf{N} \approx 1000$).

The simulation compute-times and speedups compared to the serial MEX integrators are shown in Fig. 7.11 and Table 7.3. We see that, once again, the higher dimensionality has reduced the speedup performance of the GPU codes. However, the decrease in speedup is less (in percent) than the decrease from the one- to the two-dimensional codes. A possible explanation is that the three-dimensional codes have even more boundary cells in the CUDA block ($O(N^2)$) than the two-dimensional case, and those cells require even more global memory accesses (as shown in Sec. 7.2.4, the 2SHOC scheme required up to 7 accesses for the corner cells). The additional number of grid boundary cells can also be a factor when using the MSD boundary condition as was explained in Sec. 7.3.3.

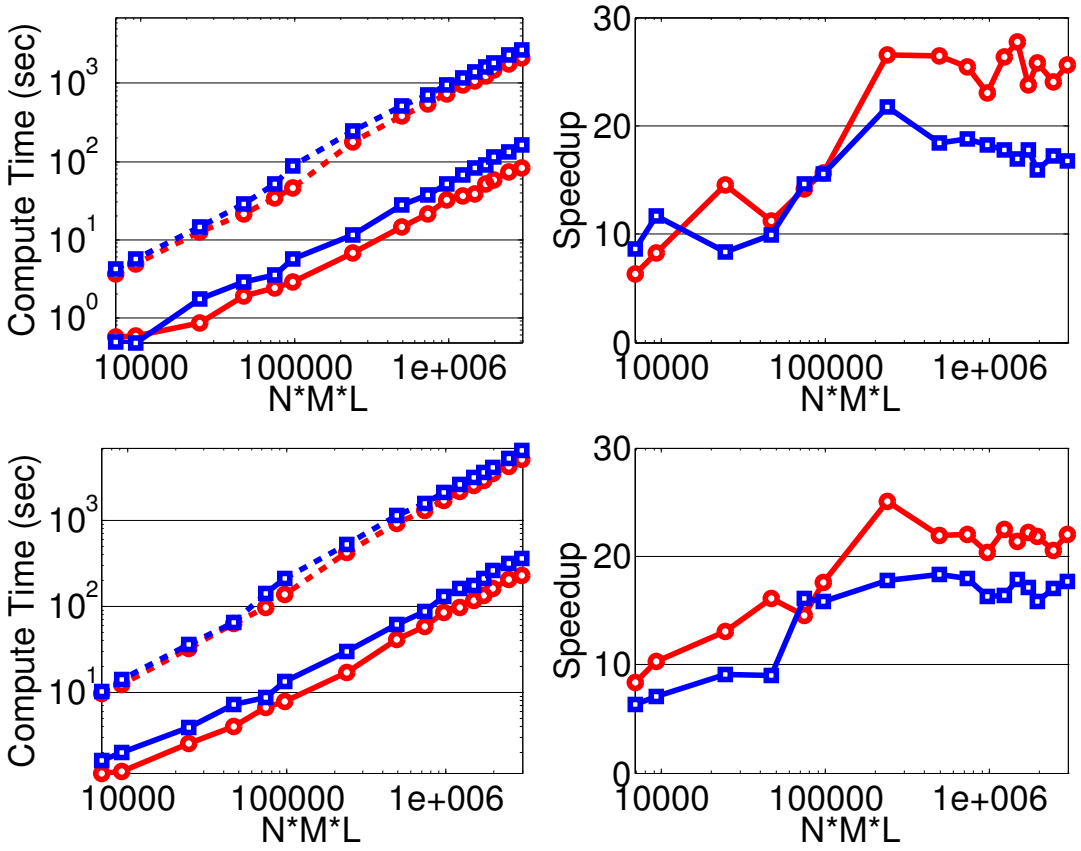


Figure 7.11. Computation times (left) and speedups (right) of the three-dimensional NLSEmagic CUDA MEX integrators (solid) versus the serial MEX integrators (dashed) for simulating the approximate dark vortex ring solution described in Sec. 6.1 with an end-time of $t = 50$. The results are shown for both the RK4+CD scheme (top) and the RK4+2SHOC scheme (bottom) using both single (red circles) and double (blue squares) precision.

Table 7.3. Subset of the three-dimensional timing results from Fig. 7.11. The results are shown for the RK4+CD scheme in single (s) precision (best results) and for the RK4+2SHOC in double (d) precision (weakest results).

	GPU Time (sec)		CPU Time (sec)		SPEEDUP	
$N * M * L$	CD (s)	2SHOC (d)	CD (s)	2SHOC (d)	CD (s)	2SHOC (d)
9261	0.59	2.00	4.85	14.17	8.29	7.10
97336	2.90	13.34	45.43	211.23	15.65	15.84
970299	31.85	131.74	733.46	2146.78	23.03	16.30
2985984	82.14	365.47	2106.77	6460.61	25.65	17.68

Despite the reduction in performance compared to the two-dimensional results, the three-dimensional codes still exhibit very good speedups, especially considering the cost and portability of the GPU card. For example, in the RK4+CD test of $\mathbf{N} = 149^3 = 2985984$, the GPU code took about 1 minute 20 seconds, while the serial MEX code took over 34 minutes. Based on the MEX speedups of Sec. 6.3.1, the equivalent MATLAB script code would take over 5 *hours* to complete the same simulation. Also, in three-dimensional problems, even moderately resolved solutions require very large total grid sizes. Therefore, the larger speedups will be the most common in practical applications.

Like in the one- and two-dimensional cases, we show the timing results of Fig. 7.11 displayed as the number of RK4 steps per second in Fig. 7.12.

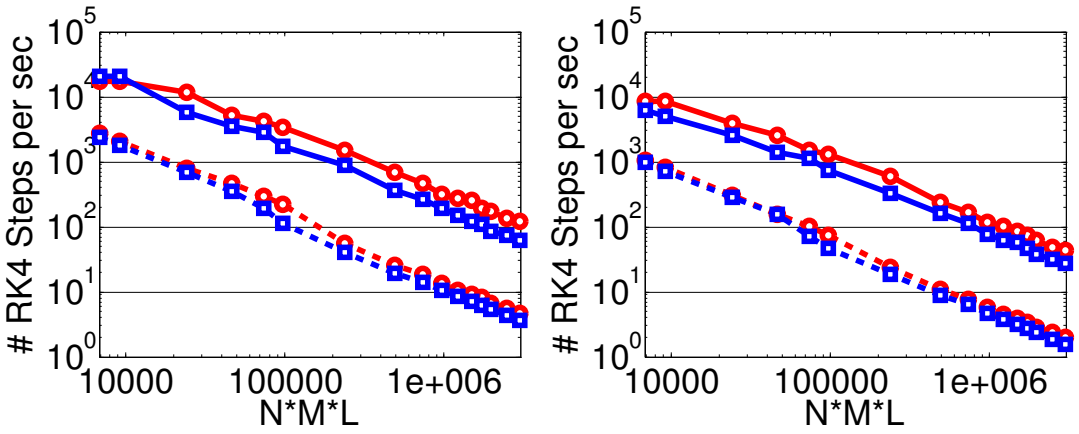


Figure 7.12. The number of RK4 time-steps per second for the simulations of Fig. 7.11 for the RK4+CD (left) and RK4+2SHOC (right) schemes. The additional figure properties are the same as those in Fig. 7.11.

All the speedup tests presented in this chapter were on grids which were nearly equal-sized in each dimension (i.e. $N \times N$ or $N \times N \times N$). Depending on the sizes, the block structure could be more or less optimized in terms of filling the edge blocks with more or less capacity. This explains why the speedups in two dimensions and (all the more so in three dimensions) are not smooth over changes in $\mathbf{N}$, but rather fluctuate. Therefore,

certain specific grid sizes have the potential to be more efficient. As an example, we have simulated a vortex ring solution in a grid with dimensions $87 \times 87 \times 203 \approx 1,500,000$ for an end-time of $t = 100$ and time-step size of $k = 0.03$ (yielding a chunk-size of only 56 which is on the low end of the efficiency requirements from Sec. 7.3.1) with the RK4+2SHOC scheme and the MSD boundary condition in double precision, and saw a speedup of over 26. This is much higher than the comparable speedup shown in Fig. 7.11 for the same total grid size and scheme. Thus, the actual speedup in any given problem may be higher (or admittedly, lower) than those shown in Fig. 7.11.

CHAPTER 8

CODE IMPLEMENTATIONS: NLSEMAGIC SOFTWARE PACKAGE

As mentioned in Chapter 1, producing distributable code packages is a way to both ensure reproducibility of research results as well as to help facilitate further research in the field. In this light, all the codes described in this dissertation are packaged together and collectively named NLSEmagic: Nonlinear Schrödinger Equation Multidimensional MATLAB-based GPU-accelerated Integrators using Compact high-order schemes and freely distributed.

The basic NLSEmagic code package contains all the serial and CUDA MEX integrators described in Chapters 6 and 7. In addition, the package contains MATLAB script driver codes which are used as examples of how to use the integrators. These scripts are called `NLSEmagic1D.m`, `NLSEmagic2D.m`, and `NLSEmagic3D.m`, and include the files `makeNLSEmagic1D.m`, `makeNLSEmagic2D.m`, and `makeNLSEmagic3D.m` which contain the commands to compile the integrators (see Appendix J for details). In order to make setting up the codes as easy as possible, we also make available pre-compiled binaries of the serial and CUDA MEX integrators for 32- and 64-bit Windows (Linux binaries are planned to be added as well).

As mentioned above, an important reason for distributing codes is to allow others to reproduce research results. The issue of reproducible research when using numerical simulations is a topic of great concern in the scientific community [126–128]. Because many codes in use are written to run on specific hardware, and are typically not available to others, it is very difficult to validate numerical results without taking a large amount of time to reproduce an equivalent code to test the simulations. This is highly unfeasible,

especially when the codes require parallelization and large computer clusters to run. In this light, we have included what we call 'full research scripts' to the NLSEmagic package. These driver scripts contain all the code necessary to reproduce the main results computed in this dissertation. This includes the initial conditions (vortices, vortex rings, etc.), as well as code to visualize, track, analyze, and save images and movies of the simulations. They therefore also contain the script integrators used in the MEX speedup calculations of Sec. 6.3.1, which can be used to develop new numerical methods. The full research driver scripts are called `NLSE1D.m`, `NLSE2D.m`, and `NLSE3D.m`, and rely on other script codes which are also included in the download package (see Sec. 11.3 for details on the use of the scripts to reproduce the results of the dissertation).

The full-research and basic driver scripts of NLSEmagic make use of some freely available third-party MATLAB codes. One such code is called `vol3D` [129] which is used to produce the volumetric renderings of the three-dimensional NLSE simulations. Another code included is `nsoli` [130] which is a Newton-Krylov nonlinear equations solver used to find numerically exact vortices and vortex rings (see Chapter 9 for details). Finally, a code called `ezyfit` [131] is used in Chapters 9 and 10 to perform simple least-squares curve fitting to formulate models of the numerical results.

The NLSEmagic code package is distributed free of charge at `http://www.nlsemagic.com`. The current website is shown in Fig. 8.1. As of this writing, the website has received over 500 unique hits from over 40 countries which indicates its interest to many researchers. The code packages and website contain all the documentation necessary to setup, compile, install, and run the various codes. The installation guide and CUDA-MATLAB setup guides are reproduced in Appendices J and I.

Figure 8.1. Screen-shots of `www.nlsemagic.com` and its sub-pages, where all code packages of NLSEmagic are distributed free of charge.

CHAPTER 9

STRUCTURE AND DYNAMICS OF A SINGLE VORTEX RING

Now that the main computational tools are complete, we turn back our focus to the study of vortex rings in the NLSE. In this chapter, we describe the structure and dynamics of dark vortex rings of charge $|m| = 1$, as well as explore the possibility of realizing dark vortex rings of higher charge. A short demonstration of unstable bright vortex rings is shown as well.

9.1 UNITARY-CHARGED DARK VORTEX RINGS

Dark vortex ring solutions to the NLSE of charge $|m| = 1$ have been the exclusive focus of the large number of previous studies (see Chapter 1). This is because the unitary dark vortex rings are typically stable and therefore are the most likely to be observed in physical experiments. In this section, we explain how to formulate a numerically-exact dark vortex ring of charge $|m| = 1$, verify its transverse velocity through the tracking of direct simulations, and explore quadrupole oscillations of the rings.

9.1.1 Obtaining Numerically-Exact Vortex Rings

The first step in achieving a numerically-exact vortex ring is to first formulate a numerically-exact solution to a two-dimensional dark vortex. This is because the two-dimensional dark vortex solution is also the steady-state solution to a three-dimensional vortex line in cylindrical coordinates, and therefore is a good approximation to the two-dimensional axisymmetric cut of the vortex ring (in fact, as the radius of the vortex rings approaches infinity, the two-dimensional solution should be

exact). We therefore will use a numerically-exact two-dimensional dark vortex as the initial iterate for use with an optimization routine to find a numerically-exact vortex ring.

As shown in Sec 2.3.1, the form of a vortex solution to the NLSE is given as

$$\Psi(\rho, \phi, t) = f(\rho) \exp\left[i\left(m\,\phi + \Omega\, t\right)\right], \tag{9.1}$$

where $f(\rho)$ is a real-valued radial profile, m is the vortex charge, and Ω is the frequency (we have used ρ instead of r and ϕ instead of θ to describe the radial and azimuthal directions in order to conform to the notation in the vortex ring's coordinate system described below). When inserted into the NLSE of Eq. (1.2), Eq. (9.1) produces the steady-state ODE

$$-\left(\Omega + \frac{a\,m^2}{\rho^2}\right) f(\rho) + a\left(\frac{1}{\rho}\frac{df}{d\rho} + \frac{d^2 f}{d\rho^2}\right) + s\, f^3(\rho) = 0. \tag{9.2}$$

The real-valued radial profile $f(\rho)$ can be obtained numerically by using a nonlinear equation solver. We define $F(f(\rho), \rho)$ as

$$F(f(\rho), \rho) = -\left(\Omega + \frac{a\,m^2}{\rho^2}\right) f(\rho) + a\left(\frac{1}{\rho}\frac{df}{d\rho} + \frac{d^2 f}{d\rho^2}\right) + s\, f^3(\rho), \tag{9.3}$$

and want to minimize $||F(f(\rho), \rho)||$.

One common-used approach is to not use nonlinear solvers but instead to use imaginary time integration of the steady-state NLSE to find the solution. This method has been used in finding vortex ring solutions with success [16, 41]. However, since the basic method has difficulty finding solutions of higher-charged vortex structures, we choose to use nonlinear equation solvers.

Since Eq. (9.3) is only one-dimensional, almost any solver can be used even if they require the formulation of the explicit form of the Jacobian matrix within the algorithm (as most solvers do) [132]. However, since we will be using a nonlinear equations solver to

refine the vortex solution into the two-dimensional vortex ring axisymmetric cut, to keep the code simple, we choose to use a matrix-free method for both optimizations.

The nonlinear equation solver code utilized is called `nsoli` [130] which contains several variations of Newton-Krylov methods which are all implemented as matrix-free algorithms. A Newton-Krylov solver uses a nonlinear inexact-Newton iterated solver and within each iterate, a Krylov-subspace linear iterative solver is used to compute the Newton direction. The method we choose uses the restart generalized minimal residual method (GMRES(n)) [133] for the linear iterations, where n is the maximum number of vectors allowed in the generated Krylov subspace before a restart is initialized.

We set the maximum number of nonlinear iterations to 70, $n = 50$, and a restart limit of 2. Although `nsoli` typically uses the L2-norm of the residual as its stopping criteria, we have changed this to be the infinity-norm, as it allows one to set the same tolerance for any size grid and have the solution's accuracy be comparable.

For the initial iterate, we use the asymptotic approximate profile of Eq. (2.25) and set the stopping tolerance to be 10^{-15}. Dirichlet boundary conditions are used, where $f(0) = 0$, and the asymptotic profile of Eq. (2.25) is used to compute $f(\rho_{\text{max}})$. Since $f(\rho)$ converges to the background density very slowly (as mentioned in Chapter 4), to improve accuracy of the boundary condition, we compute $f(\rho)$ out to a distance much greater than is needed for grid of the vortex ring. Numerically-exact radial profiles of dark vortices of charge $m = 1$ and $m = 3$ are shown in Fig. 9.1.

To aid in the present discussion, the coordinate system of a vortex ring described in Fig. 2.8 is reproduced here in Fig. 9.2. The vortex ring solution is axisymmetric in the θ-direction and can therefore be represented as a two-dimensional axisymmetric cut in the r–z plane whose initial condition is represented as

$$\Psi_0 = \Psi(r, z, \theta, 0) = g(r, z) \exp[i\, m\, (\phi(r, z) - \phi_2(r, z))]. \tag{9.4}$$

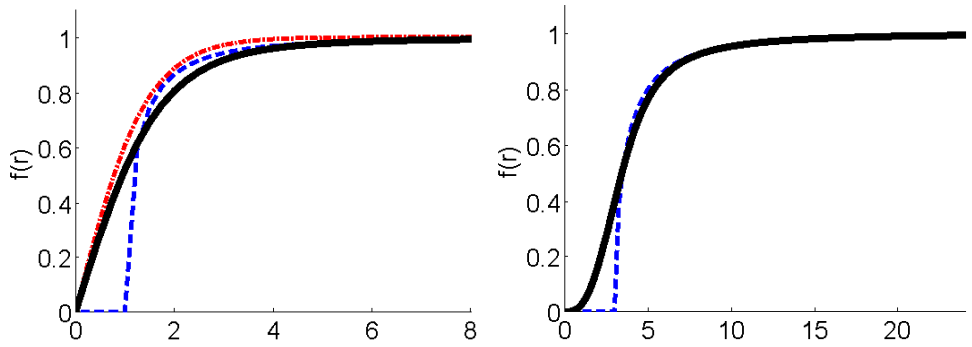


Figure 9.1. Numerically-exact two-dimensional dark vortex radial profiles (black lines) for charge $m = 1$ (left) and $m = 3$ (right). The asymptotic approximate profiles (Eq. (2.25)) used as the initial iterate to the optimization routine are shown as dashed (blue) lines. For $m = 1$, the one-dimensional dark soliton profile (dot–dashed (red) line) is shown for comparison.

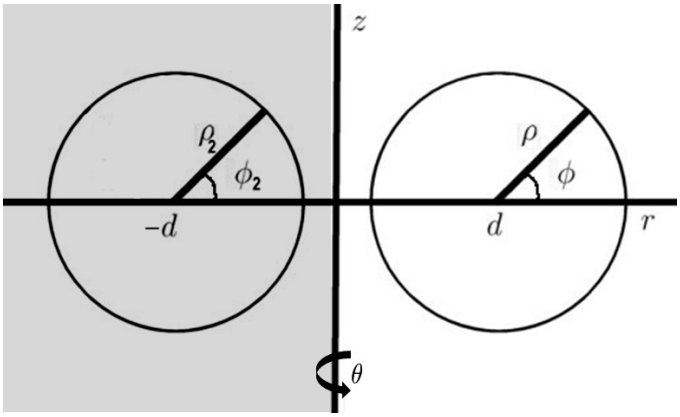


Figure 9.2. Coordinate system used to describe vortex rings. The ϕ_2 and ρ_2 variables are only used in the half plane ($r \geq 0$).

The ansatz of Eq. (9.4) takes into account much of the phase structure of the vortex ring by including the $m\,\phi_2$ term. However, it assumes the phases of the two vortex cores interfere in a purely linear way, which is not the case. Therefore, unlike the $f(\rho)$ of Eq. (9.1), $g(r, z)$ cannot be taken to be purely real. However, since the amount of additional phase information in $g(r, z)$ is assumed to be small, as an initial iterate to the numerical optimization, we use the numerically-exact real-valued $f(\rho)$ of Eq. (9.1) rotated in the ϕ direction for $g(r, z)$. This must be done through a linear interpolation since the

$\rho = \sqrt{(r-d)^2 + (z-z_0)^2}$ values of the grid to contain the vortex ring do not coincide to the equidistant ρ values used to compute the one-dimensional $f(\rho)$.

To numerically optimize Ψ_0, it is necessary to have the vortex ring solution situated in a co-moving frame at its velocity c, so that the governing equations are time-independent. Using the form of a co-moving solution to the NLSE as shown in Sec. 2.4 and Appendix B, the governing steady-state PDE for the vortex solution in the r–z plane is given by

$$F(U, r, z) = -\Omega\, U + a \left(\frac{1}{r} \frac{\partial U}{\partial r} + \frac{\partial^2 U}{\partial r^2} + \frac{\partial^2 U}{\partial z^2} \right) + s\, |U|^2\, U = 0. \tag{9.5}$$

In order to form a steady-state initial iterate, the intrinsic velocity of the vortex ring needs to be counterbalanced by adding a back-flow with a velocity equal and opposite to that of the vortex ring. Using Eq. (2.40), the form of the steady-state initial iterate becomes

$$U_0(r, z) = \Psi_0 \exp\left[-i \frac{c}{2\, a}\, z \right], \tag{9.6}$$

where c is defined by the analytical approximation of the vortex ring velocity given by Eq. (2.39).

The two-dimensional ansatz of Eq. (9.6) must be unwrapped into a one-dimensional vector in order to be used with the Newton-Krylov solver. In addition, since the numerical solver only takes real-values as input and U_0 is complex-valued, the real and imaginary parts of U_0 are separated and then concatenated to each other, forming one long vector.

Since the true values of U_0 at the boundaries is not known or able to be accurately approximated, it becomes difficult to apply proper boundary conditions to Eq. (9.5), especially due to the singularity at $r = 0$. Since the co-moving vortex ring is steady-state, it is possible to use the modulus-squared Dirichlet (MSD) boundary condition described in Chapter 4. This can be done by noting that the form of the MSD boundary condition

applied to the Laplacian of the NLSE (4.19) is not time-dependent. Therefore, the boundaries of Eq. (9.5) can be computed based on the nearest interior points which has the benefit of avoiding the singularity at $r = 0$. As discussed in Chapter 4, the expression $\nabla^2 U/U$ near the boundaries should be real-valued. In Eq. (4.19), the possible numerical addition of an imaginary part was suppressed by only taking the real part of $\nabla^2 U/U$ in the MSD boundary condition. In the present context of a numerical optimization, this error should *not* be artificially suppressed, as the lowering of that error should be part and parcel of the optimization process. Therefore the form of the MSD boundary condition used for the nonlinear equation solver is

$$\nabla^2 U_b \approx \left[\frac{\nabla^2 U_{b-1}}{U_{b-1}} + \frac{1}{a} \left(N_{b-1} - N_b \right) \right] U_b, \tag{9.7}$$

where

$$N_b = s \left| U_b \right|^2 - V_b, \qquad N_{b-1} = s \left| U_{b-1} \right|^2 - V_{b-1},$$

where b represents a boundary point and $b - 1$ a point one cell inwards in the normal direction.

Due to the increased size of the solution, the free-boundary conditions, and the fact that an arbitrary phase rotation of U_0 leaves the residuals of Eq. (9.5) unchanged, the convergence of the `nsoli` code is much more difficult to achieve than in the case of optimizing $f(\rho)$. Therefore we set our tolerance to only be 10^{-5} and note that many times the solution still does not converge and has a total residual of around 10^{-4} after the maximum number of iterations is complete. Increasing the number of iterations does decrease this error but only very slowly and is not worth the extra computational cost. However, the residual of 10^{-4} is still quite low as the initial error is around 0.1, and the numerically optimized vortex ring travels without noticeable distortions in the background density as shown in Fig. 9.3.

After the optimization of U_0 is complete, the initial condition of the vortex ring is recovered by multiplying out the co-moving back-flow, and linearly interpolating uniform r-defined z-columns of Ψ_0 on the $r = \sqrt{(x-x_0)^2 + (y-y_0)^2}$ values of the three-dimensional grid to obtain

$$\Psi(r, z, \theta, 0) = \Psi_0 = U_0 \exp\left[i \frac{c}{2\,a}\, z\right], \tag{9.8}$$

In order to ensure that Ψ_0 is formulated for all the required values of r, the size of the optimized r–z plane is chosen to ensure complete coverage of the three-dimensional grid. The resulting vortex ring solution grid can then be rotated by rotation matrices to formulated rings oriented at different angles (once again ensuring the optimized grid is large enough to cover all require values of the rotated grid). The solution can then be modified to include back-flow velocities and phase shifts as desired. Some examples of optimized vortex rings are shown in Fig. 9.3. It is observed that there is no visual distortion in the structure of the vortex ring over the duration of the simulation.

In optimizing the vortex rings we used the steady-state PDE governing U of Eq. (9.5) instead of the PDEs governing $g(r, z)$ directly given by Eqs. (2.44) or (2.46). This can potentially allow the solution to 'drift' during optimization away from the prescribed vortex ring diameter d since there is nothing constraining the solution's vortex ring to have a specific diameter. However, as the initial iterate used is quite close to the true solution, in practice, the final vortex ring always has the radius initially chosen in the initial iterate. The reason for avoiding using the PDEs of Eqs. (2.44) or (2.46) is that they contain an additional singularity at the vortex ring core diameter ($r = d$, $z = z_0$), which unlike the singularity at $r = 0$, cannot in general be avoided since it is in the middle of the grid. Handling such singularities properly is not trivial [134] and in our tests, using the PDEs governing $g(r, z)$ do not yield as refined solution as using Eq. (9.5).

In formulating the initial iterate of Eq. (9.4), we used ϕ_2 coordinate to describe the phase interference of the two sides of the vortex ring. As mentioned, this makes the initial

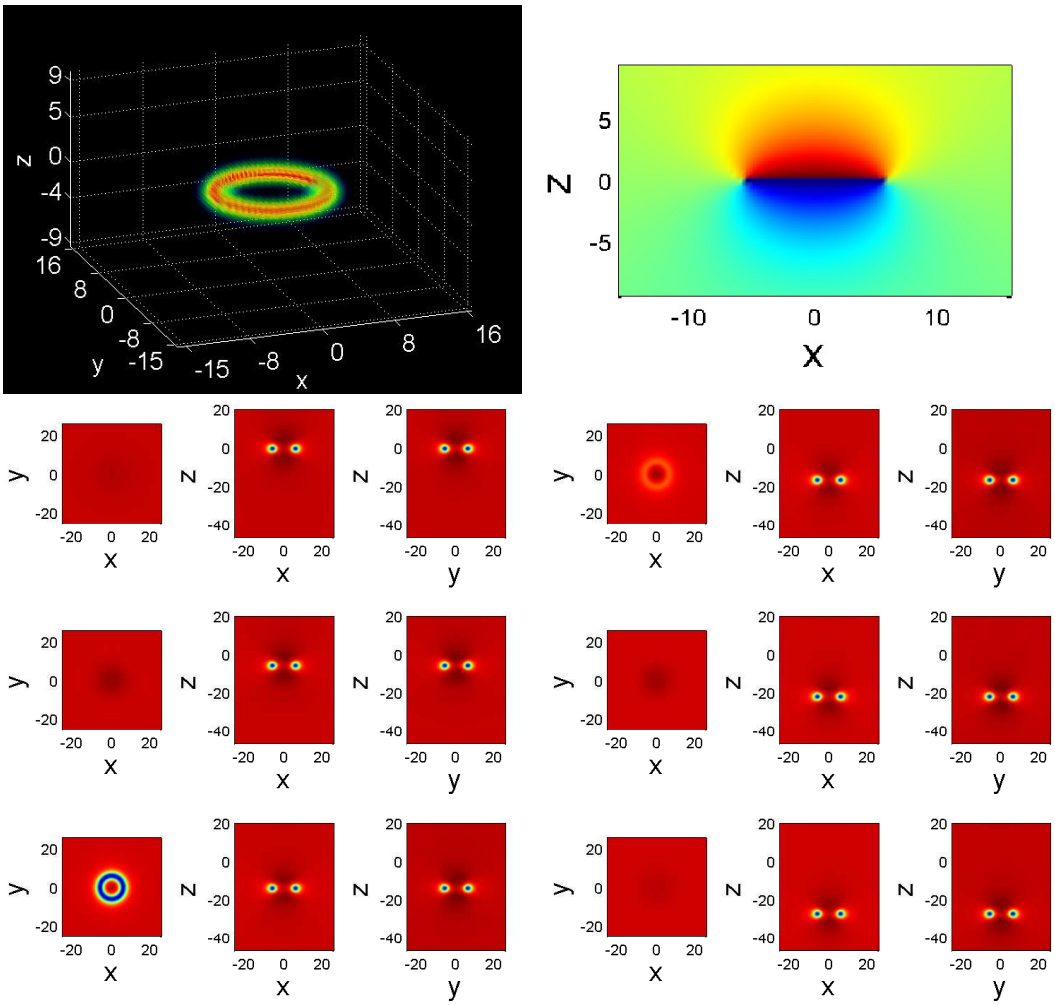


Figure 9.3. Top left: Volumetric rendering of the inverse-density of a numerically exact vortex ring of charge $m = 1$ and radius $d = 10$. Top right: The phase of the same vortex ring in a y–z cut at $x = 0$. Bottom: y–z cuts of a vortex ring of radius $d = 6$ traveling at its transverse velocity until time $t = 50$ (top to bottom, left to right). The simulation uses the NLSE parameters $a = 1$, $\Omega = -1$, $s = -1$, and numerical parameters $h = 1/2$ and $\mathbf{k} = 0.026$.

iterate's phase much closer to the true solution than compared to using $\exp(i\, m\phi)$ alone. In Fig. 9.4, we show the initial iterate phase when using only $m\phi$ and when using $m\phi - m\phi_2$ compared to the numerically exact phase profile. It is clear that using

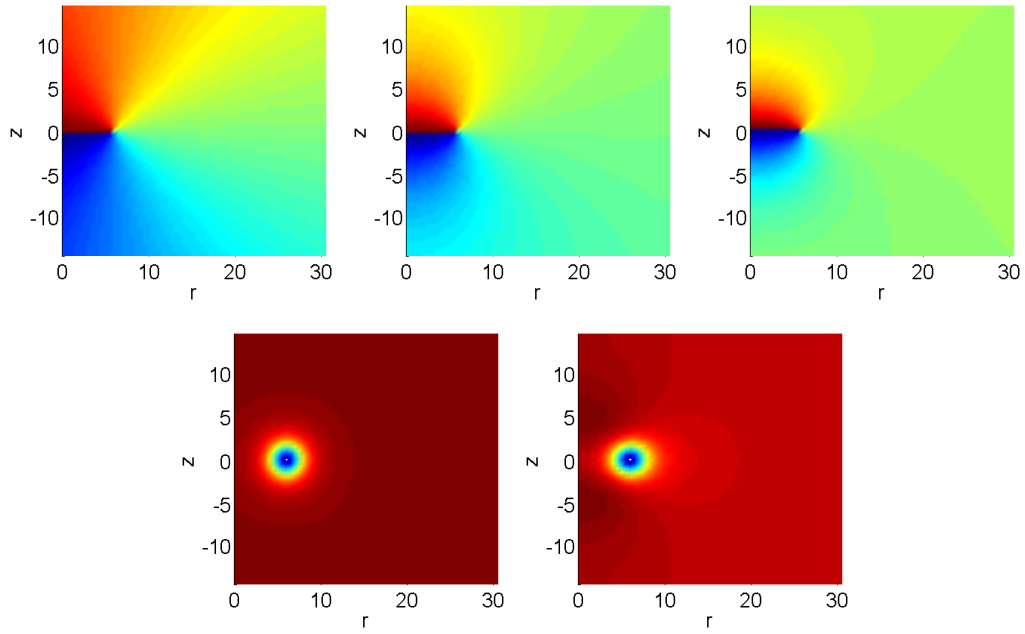


Figure 9.4. Top row left to right: The phase of the initial axisymmetric vortex ring cut iterate utilizing $m\Phi$ and $m\phi - m\phi_2$ in Eq. (9.4) and the optimized numerically-exact solution. Bottom row: The density $|\Psi|^2$ of the initial iterates and the numerically-exact solution.

$m\phi - m\phi_2$ is much closer to the true phase structure of the vortex ring than using $m\phi$.

9.1.2 Translational Velocity

Although the translational velocity of vortex rings has been studied analytically and has been numerically confirmed (see Chapter 1), the confirmation of the asymptotic approximate velocity has not been quantified. In this section we track vortex rings of various radii and directly measure their translational velocity. This velocity is then compared to that given by Eq. (2.39).

In order to compute the velocity of the ring, it is necessary to be able to track its position during the simulation. Since the vortex ring is only traveling in the z direction, it

can be tracked by simply noting the z-position of the minimum value of $|\Psi|^2$. However, since it will be necessary to track more diverse motion of vortex rings in further sections, a more general tracking system is used.

A two-dimensional y–z cut of the computational grid at $x = 0$ is extracted which allows for easy tracking of the z and y positions of the vortex ring's cut. This obviously does not capture all the motion of the vortex ring, but for many scenarios (such as those in the current chapter and Chapter 10) the tracking of the y–z cut is sufficient.

Although the discretization of the vortex ring solution can have a fairly coarse spatial step-size and still be very accurate (due to the global nature of the dark vortex rings, as well as the high-order scheme used), the coarse grid can make tracking the position of the vortex rings difficult. A center-of-mass numerical integral over a defined local search window would make a smoother position track, however since the vortex rings are dark, this cannot be done directly. Instead, the modulus of the vorticity in the cut is used as it is a positive narrow spike at the location of the vortex ring. As such, a center-of-mass calculation can be easily done which allows the vortex to be tracked smoothly.

To measure the transverse velocity, the vortex ring's center z-position is recorded over the run and a linear least-squares fit to the results is performed. In Chapter 4 we found that the computational grid boundaries should be at least a distance of 20 from the vortex core in all directions for accurate velocity results. In Fig. 9.5 we show the results of integrating the vortex ring to an end time of $t = 50$ for radii $d = 2 \to 10$ and compare the velocities to Eq. (2.39). It is seen that the direct velocity results match the analytical approximation very accurately, never deviating more than 1.5%. This shows that the value of the asymptotic velocity equation of Eq. (2.39) is very accurate even for rings with relatively small radii.

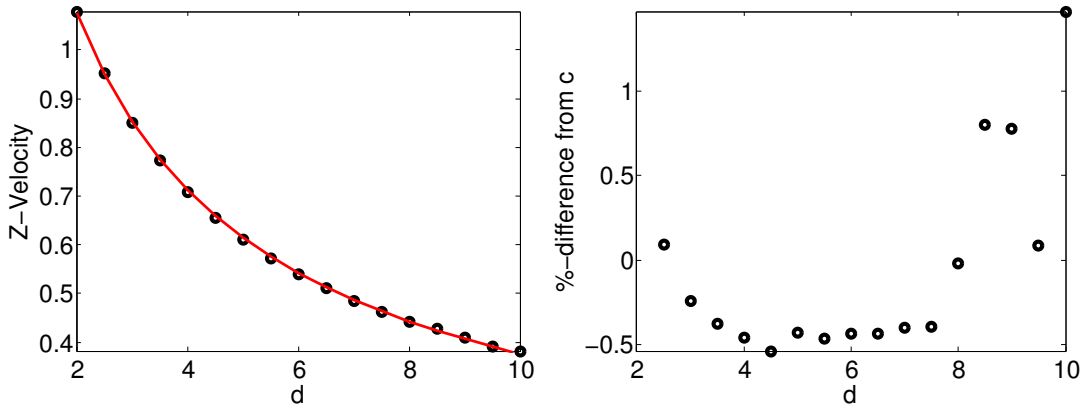


Figure 9.5. Left: The velocity of dark vortex rings of charge $m = 1$ for radii $d = 2, 2.5, ..., 10$. The dots are the velocities computed from direct simulation averaged over a $t = 50$ run, while the line is the predicted velocity of Eq. (2.39). Right: Percent difference between the measured and predicted values of the velocity. The NLSE parameters used are $a = 1$, $s = -1$, and $\Omega = -1$, while the numerical parameters are $h = 0.5$ and $\mathbf{k} = 0.025$.

9.1.3 Quadrupole Oscillations

When a vortex ring is perturbed, it can oscillate and affect its dynamics considerably. A study of vortex rings perturbed by planar and helical Kelvin modes was performed in Ref. [135]. It was found that the Kelvin modes can slow down the vortex rings, even causing them to stop or travel backwards. The authors there showed results of the slowdown for various modes, all for vortex rings of radius $d = 25$.

The lowest order Kelvin mode is a planar mode of two which causes the vortex ring to undergo quadrupole oscillations (see Fig. 9.6 for an example). We see that the two-dimensional cut of the oscillating vortex ring does not have a constant transverse velocity over time, however an averaged constant velocity can be obtained which is a good indicator of the true slow-down of the vortex ring caused by the perturbation.

Quadrupole oscillations are interesting, independent of a full Kelvin mode, study because they are typically the most dominant mode perturbed when vortex rings merge and separate. It is therefore of interest to see how the quadrupole mode behaves in more detail. Specifically, we would like to know how the amplitude and frequency are dependent on each other for various values of the vortex ring's radius d.

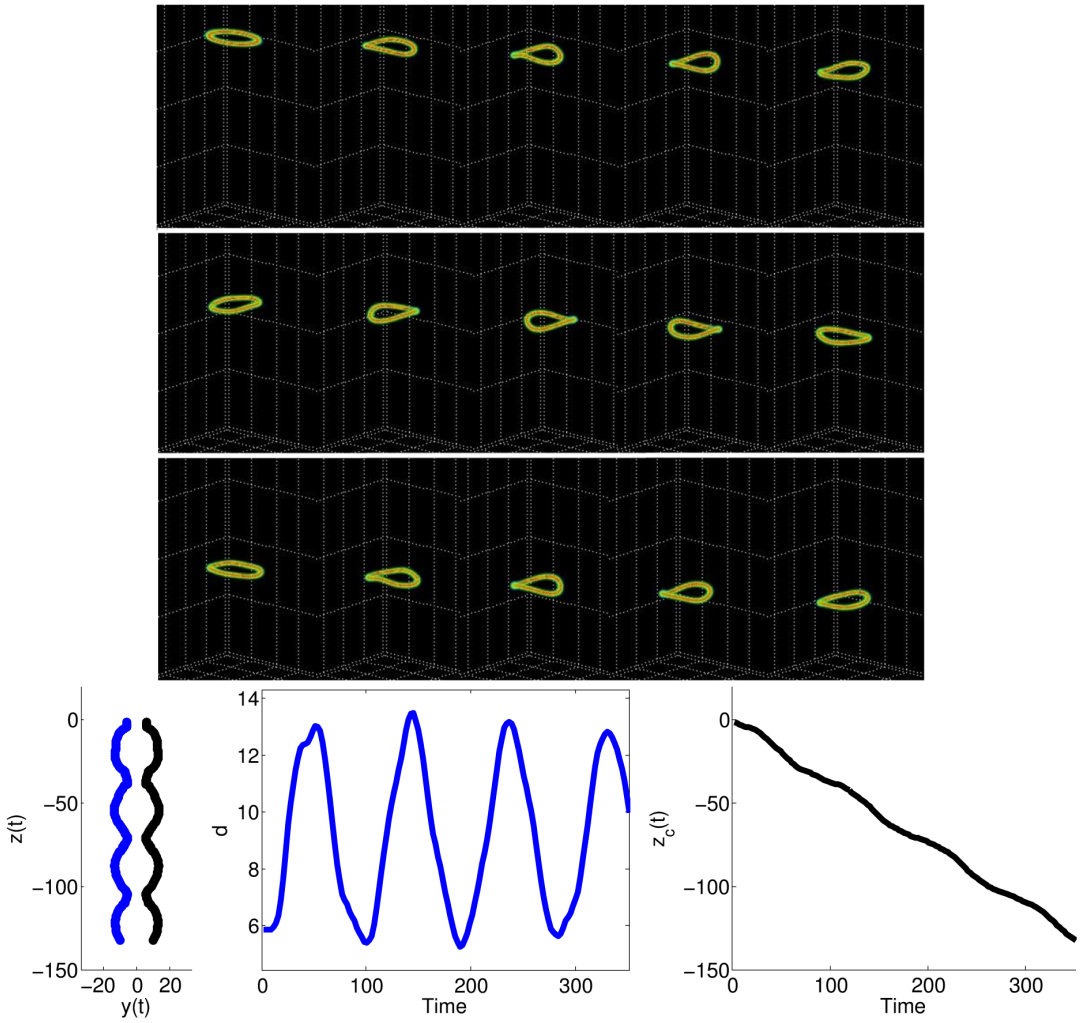


Figure 9.6. Example of quadrupole oscillations of a dark vortex ring of radius $d = 6$. Top: Snapshots of a simulation where the mode is perturbed with an amplitude equal to the vortex ring's radius ($A/d = 2$) taken from time $t = 0$ to $t = 140$. Bottom: Tracking results in the y–z plane at $x = 0$ of the same vortex ring for 350 time units. Shown left to right are the traced position, radius versus time, and centered z-position versus time. The NLSE parameters are the same as in Fig. 9.5 and the numerical parameters are $h = 0.5$ and $k = 0.025$ (top), and $h = 2/3$ and $k = 0.046$ (bottom).

We also record the velocity of the ring versus the mode amplitude. Although this was previously studied in Ref. [135], the results shown had very large error-bars and it is unclear how the velocity actually behaves depending on the mode amplitude. The large error-bars there were understandable since they used a vortex ring of radius $d = 25$ which is quite large, and thus its velocity is slow and hard to track (this can be seen in our velocity results of Fig. 9.5 where the measured velocity for the larger rings deviated from the analytical approximation more than for the smaller rings). Additionally, Ref. [135] did not use numerically exact vortex ring solutions, but rather an analytical approximation. Therefore, since we will be using smaller vortex ring radii whose velocities are easier to track, a higher maximum mode amplitude, numerically-exact vortex rings (which are then perturbed), and will show the results for a few different radii, the revisiting of the velocity versus amplitude is useful as a supplement to Ref. [135].

To imprint the perturbed mode onto the vortex ring, we define the r coordinates in our interpolation of the axisymmetric cut of the vortex ring to be an ellipsoid in x and y instead of a perfect circle (see Sec. 9.1.1). We define the amplitude A of the vortex ring's perturbation as the distance of the semi-major axis of the ellipse. Although stretching the vortex ring to imprint the perturbation alters the structure of the vortex ring's core, the stretching is small and the vortex rings very shortly settle to the quadrupole oscillations with the correct core density. However, this may make the settled oscillatory solution have a slightly different d and A than originally chosen. Therefore, for accuracy, we record the maximum and minimum radii during the simulation after it has settled and use those for the values of A and d in our results.

To track the vortex, we use a y-z cut of the solution as in Sec. 9.1.2, and measure the radius and position of the oscillatory vortex ring. The average velocity is then computed with a least-squares linear fit to the position data, while the frequency is determined from the radius versus time data. In order to ensure accurate results, we make the planar grid dimensions large enough so that the distance from the vortex core to the

grid boundary is 20 irrespective of the amplitude of the perturbed mode. Also, the simulations are performed for long enough time to capture multiple periods in order to be able to capture the settled oscillation. Our results are depicted in Fig. 9.7. We see that the

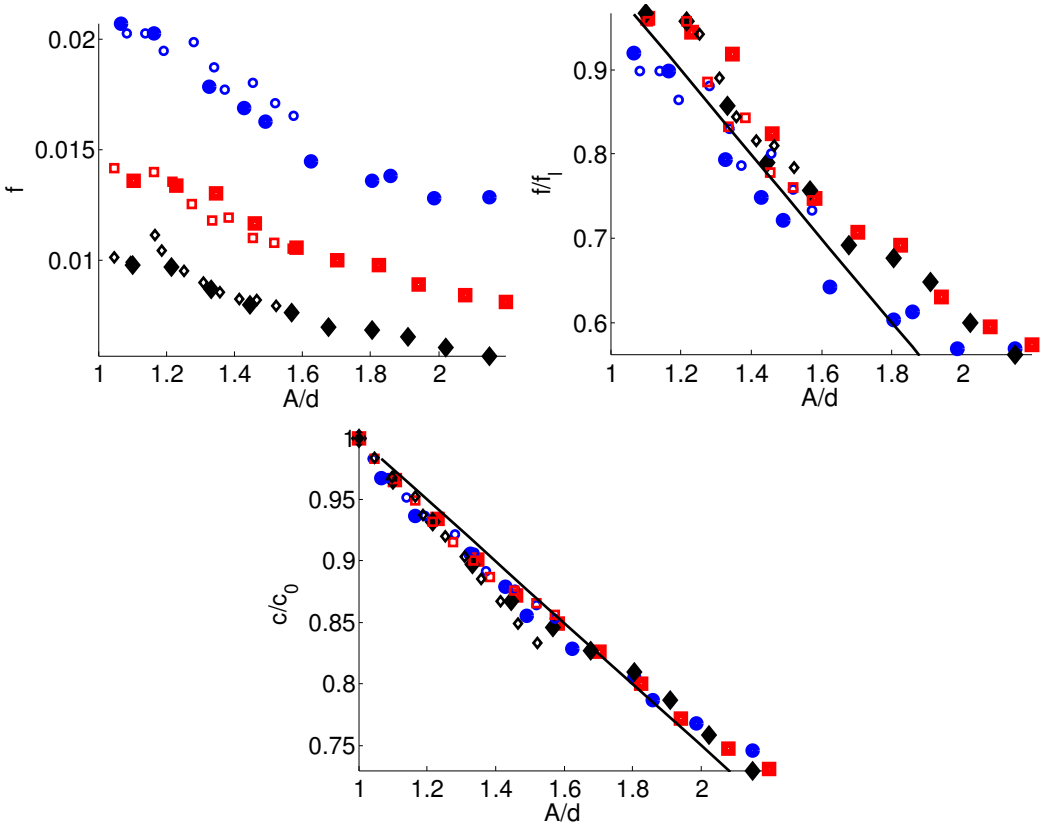


Figure 9.7. Top left: Frequency versus amplitude of vortex rings of radius $d = 6, 8, 10$ ((blue) circles, (red) squares, and (black) diamonds respectively) undergoing quadrupole oscillations. Top right: Same as left plot but the frequencies are divided by each ring's linear frequency (taken as the frequency with $A/d = 1.05$) The solid line is the approximation given by Eq. (9.9). Bottom: Average transverse velocity versus amplitude of the rings (the solid line is the approximation given by Eq. (9.10)). In all plots the smaller markers are from a run with an end-time of $t = 200$ with $h = 2/3$ and $\mathrm{k} = 0.045$ while the large markers are for simulations with an end-time of $t = 300$ with $h = 1$ and $\mathrm{k} = 0.1$. The NLSE parameters are the same as in Fig. 9.5.

frequencies start out constant and then decrease as the amplitude is increased. This is indicative of the existence of a linear frequency for small perturbations which is modified by nonlinear effects as A/d is increased. The relationship of the nonlinear frequency

variation versus the amplitude is nearly linear (up to $A/d = 2$) yielding the approximation

$$f(A/d) \approx f_l(d) \left[1 + \frac{1}{2}\left(1 - \frac{A}{d}\right)\right], \tag{9.9}$$

where f_l is the vortex ring's linear frequency of oscillation. From the $A/d = 0.05$ simulations in Fig. 9.7 for $d = 6, 8,$ and10, these linear frequencies are approximately

d	6	8	10
$f_l(d)$	0.02252	0.01416	0.01012

.

The relationship between the slowdown of the vortex ring's velocity versus the amplitude of the perturbed mode is also nearly linear yielding the approximation

$$c(A/d) \approx c_0(d) \left[1 + \frac{1}{4}\left(1 - \frac{A}{d}\right)\right], \tag{9.10}$$

where c is the perturbed ring's velocity, and c_0 is the equivalent unperturbed ring's velocity (given by Eq. (2.39)). Using Eq. (9.10) and taking advantage of the MSD boundary condition's ability to simulate co-moving back-flows (see Chapter 4), we can set up a oscillatory vortex ring amidst a back-flow computed by Eq. (9.10) to be able to simulate the quadrupole oscillation for very long simulation times while not being hindered by an excessively large grid. In Fig. 9.8 we show snapshots of such a simulation for a vortex ring of radius $d = 5$ and perturbation amplitude $A/d = 2$. We see that not only is the velocity given by Eq. (9.10) accurate (since the vortex rings stays near the center of the grid), but that even very large amplitude quadrupole oscillations of a dark vortex ring are robust and stable over long periods of time.

9.2 BRIGHT VORTEX RINGS

In a bright vortex ring, the density in the ρ direction trails off to zero rapidly in both directions from r_c. Therefore, the one-dimensional numerically-exact density profile of a bright vortex should be very close to the true vortex ring profile, and no

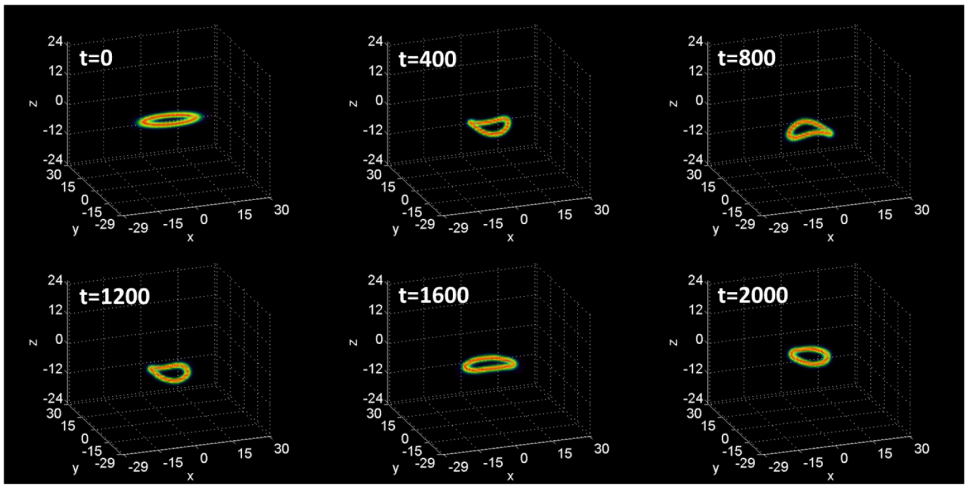


Figure 9.8. Example of a dark vortex ring with radius $d = 5$ undergoing quadrupole oscillations amidst a co-moving back-flow velocity given by Eq. (9.10) for a total time of $t = 2000$. The time of each frame is indicated. The ring is initially perturbed with a large $A/d = 2$ perturbation. The NLSE parameters are the same as in Fig. 9.5 and the numerical parameters are $h = 2/3$ and $k = 0.0458$.

two-dimensional numerical optimization in the r-z plane is necessary for an accurate solution.

The most prominent dynamics of a bright vortex ring is that, like the bright two-dimensional vortices [13], it is unstable and collapses. Some examples of this collapse are shown in Fig. 9.9 for bright vortex rings of charge $m = 1$ (with radii $d = 5$ and $d = 20$) and $m = 3$ (with radii $d = 10$ and $d = 20$). We see that in each case the vortex ring collapses into singularities. The rings collapse too quickly to determine if they have any transverse velocity or not. For small values of d, the core of the vortex ring collapses and the entire ring falls into a single singularity, while for larger values of d, the ring exhibits the expected azimuthal instability along ϕ creating a collapsing thin ring, but then also breaks up into singularities along the θ direction as well. As the number of such singularities is typically four for most of our simulations, it is suspected that the break-up in the θ direction may be effected by the choice of a Cartesian grid and different results could possibly be obtained by running the simulations on a cylindrical computational grid (such simulations are outside of the present scope). Along the core radial direction (ρ) the

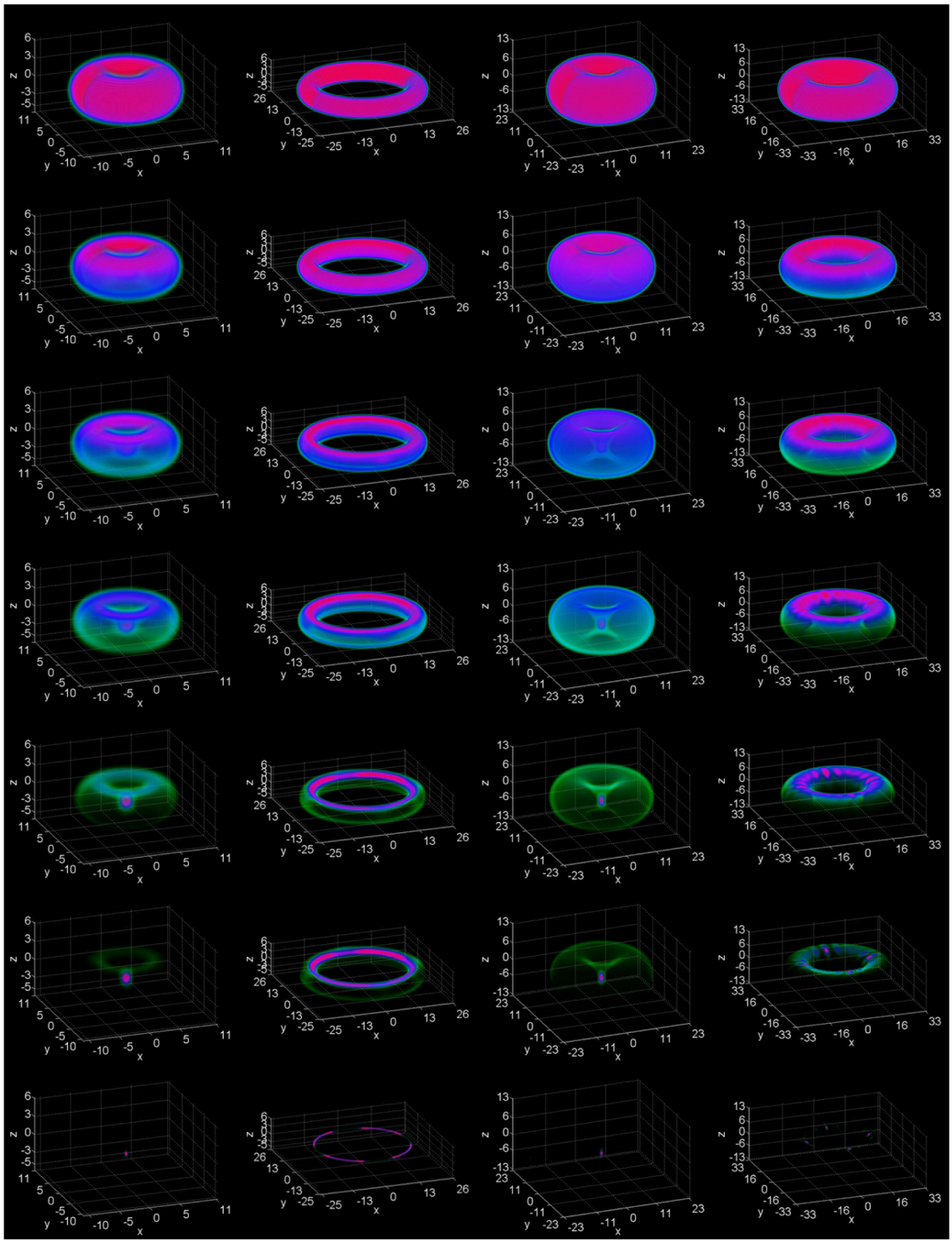


Figure 9.9. Bright vortex rings undergoing collapse. The rings are shown with charge $m = 1$ with radii $d = 5$ and $d = 20$ and for charge $m = 3$ with radii $d = 10$ and $d = 20$ (left to right). Each simulation is run until the time of collapse which is $t \approx 2.1, 5, 2.2, 7.2$ (left to right). The NLSE parameters in all simulations are $a = 1$, $s = 1$, $\Omega = 1/3$, while the numerical parameters are $h = 1/4$ and $\mathbf{k} = 0.0055$.

azimuthal instability of mode 1 seems to be the most unstable mode even in the case of $m = 3$ where, in the two-dimensional vortices, has mode 5 or 6 as the most unstable (see Chapter 2). This may be due to the fact that the density in the inner part of the ring is more closely grouped as the outer ring and therefore always perturbed the mode 1 more than the others.

9.3 HIGH-CHARGE DARK VORTEX RINGS

As was discussed in Sec. 2.3.3, two-dimensional dark vortex solutions of charge $|m| > 1$ exist but are unstable. The instability is weak for very small perturbations, but for larger perturbations the higher-charged vortices quickly split up into unitary-charge vortices in a tightly-packed cluster. It was also shown that two such vortices interacting with each other causes large perturbations in the vortices resulting in very rapid break-up of the cores. Since a vortex ring is analogous to two vortices of opposite charge traveling in parallel, the same situation is expected to occur.

In Fig. 9.10, we show the results of integrating unoptimized vortex rings of charge $m = 2$ and $m = 5$. For $m = 2$ we show the results for radii $d = 6$ and $d = 15$. We see that, as expected, the rings quickly break up into vortex rings of charge $m = 1$. However, there is a large difference between the small and large radii used for the ring of charge $m = 2$. For small radii, the ring breaks up very quickly and viably into two rings, which then proceed to interact in peculiar ways. After a while, a vortex ring is nucleated in the center of the double-ring formation which then collides with the bottom ring and either destroys it leaving one ring, or interacts with it forming four rings which travel away in symmetric transverse directions. The exploration of this interaction is beyond the current scope and left as an open problem. For large radii, the vortex ring splits up into two rings rapidly but the rings are very close to each other to the point that in the modulus-squared volumetric renderings, their distinction is not perceivable. Therefore, in Fig. 9.10, we include volumetric renderings of the vorticity of the solution in which the double-ring structure is more apparent. The charge $m = 5$ vortex ring immediately breaks up into five

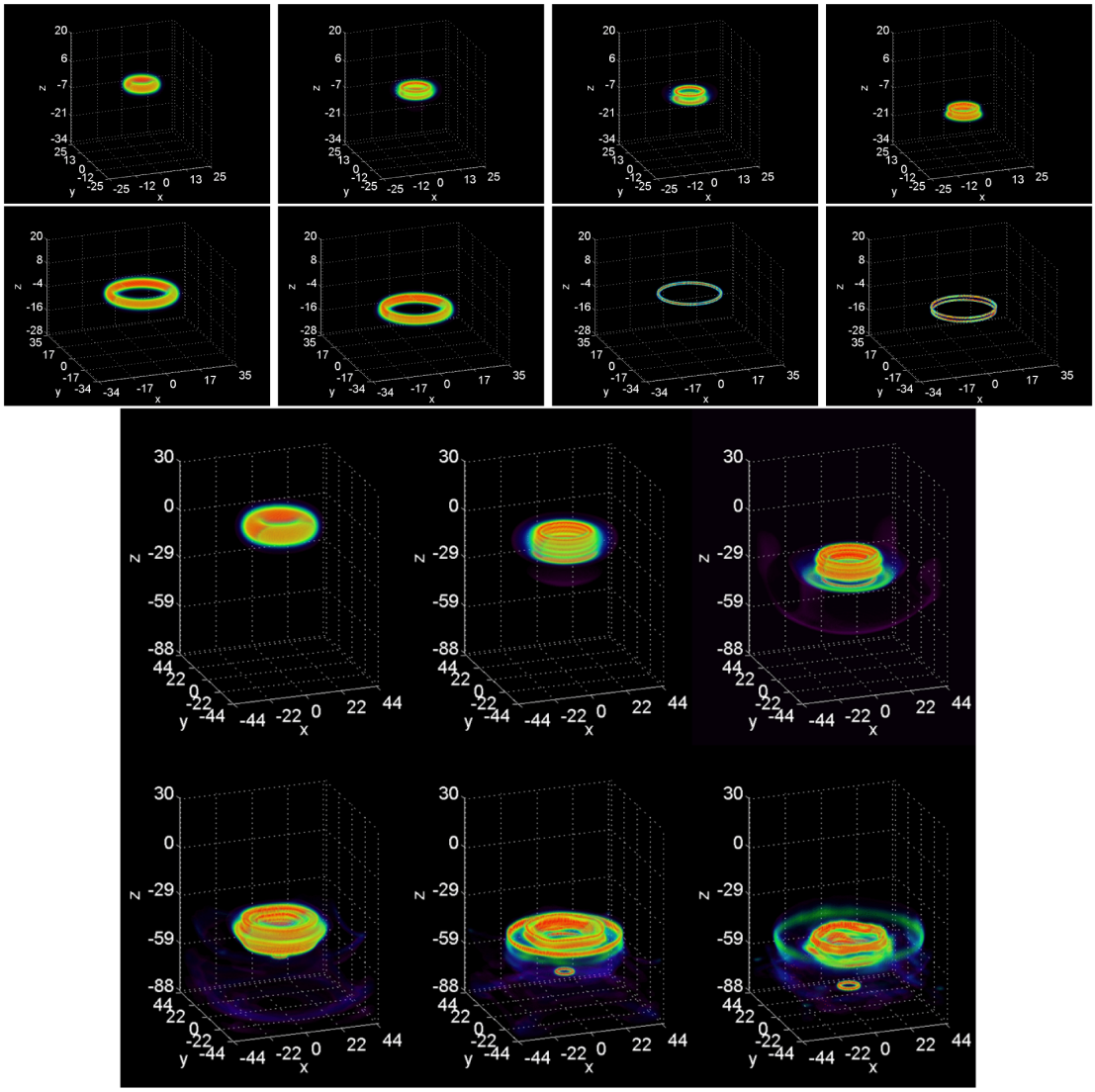


Figure 9.10. Break-up of unoptimized dark vortex rings. Top to bottom: (a) Modulus-squared of a charge $m = 2$ ring with radius $d = 6$ at times $t = 0, 5, 8, 18$, (b) modulus-squared (left two panels) and vorticity (right two panels) of a charge $m = 2$ ring with radius $d = 15$ at time $t = 0$ and $t = 20$, (c) a charge $m = 5$ ring with radius $d = 20$ at times $t = 0, 16, 36, 56, 66, 76$. The NLSE parameters are the same as in Fig. 9.5 and the numerical parameters are $h = 1/2$ and $h = 2/3$ for the charge $m = 2$ and $m = 5$ simulations respectively and k $= 0.026$.

single-charge vortex rings which then undergo multiple dynamical interactions including ring destruction, nucleation, and leapfrogging (see Sec. 10.2.1).

In order to better study the higher-charged vortex rings, it is desirable to run the optimization routines used in Sec. 9.1.1 to obtain numerically-exact solutions. We compute the velocities of the proposed high-charge vortex rings using Eq. (2.39) with the $L_0(m)$ values shown in Table. 2.1. Through much testing, we have found that for charge $m = 2$, the optimization routines produce a double-ring solution as was observed in the simulations of Fig. 9.10 after the initial breakup of the vortex rings. As the radii of the vortex rings is increased, the z distance between the two rings decreases (which is understandable as a vortex ring of infinite radius is equivalent to a vortex line where, as shown in Sec. 2.3.3, dark vortices of higher-charge do exist).

To study the long-time behavior of the double-ring solution, we place the solution into a co-moving back-flow and observe its behavior. It is found that for a considerable amount of time, the double-ring solution remains steady-state. Later, the rings start to exhibit planar modes of perturbations but remain in their orientation. After considerable time, the perturbations move the rings from their orientation and the rings undergo leapfrogging motion (see Sec. 10.2.1). Due to the large perturbation of the leapfrogging rings, the rings eventually entangle and either separate or merge. An example of the later case is shown in Fig. 9.11. We note that although the sought-after higher-charged ring results in a co-moving multi-ring solution, the velocity predictions of Eq. (2.39) are still valid for the combined ring solution, even though each unitary-charged ring on their own would have yielded a different velocity. This interesting result has been confirmed by numerically simulating *unoptimized* high-charge vortex rings amidst a back-flow velocity computed by Eq. (2.39). It was found that the rings remained near the center of the computational grid, while in a co-moving back-flow of the velocity of a single-charged ring, the multi-ring solutions hit the boundaries.

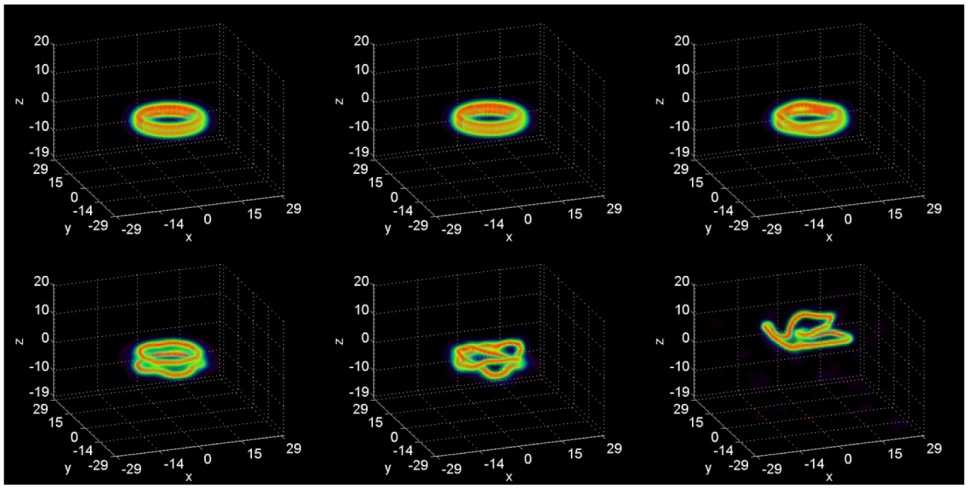


Figure 9.11. Break-up of an optimized dark vortex 'double-ring' of charge $m = 2$ and radius $d = 10$. The double-ring structure is shown from left-to-right, top-to-bottom, at times $t = 0, 140, 350, 1530, 1830, 2000$. The two rings are stable for a while, then exhibit oscillations leading to leapfrogging, and finally to entanglement and merging. The NLSE parameters are the same as in Fig. 9.5 and the numerical parameters are $h = 1/2$ and $\mathrm{k} = 0.026$.

Attempting to numerically optimize vortex rings of charge $|m| > 2$ (which like in the $|m| = 2$ case, produce a series of unitary-charges stacked rings) can be exceedingly difficult for smaller radii as the initial ansatz of a high-charge single ring is far from the true multi-ring solution. As in the $|m| = 2$ case, increasing the radii yields multi-ring solutions which get closer to a single ring solution as $d \to \infty$. Optimized multi-ring solutions for charges $m = 2, 3, 4$ and radii $d = 8, 16, 32$ are shown in Fig. 9.12. Baring any flaw in the optimization method, we can safely say that a family of 'high-charge' vortex rings do exist, but consist of co-moving multi-unitary-charged ring solutions whose vertical separation distance between rings grows as d decreases. The limiting case (as $d \to \infty$) is an infinite radius single multi-charged ring equivalent to the multi-charged dark vortex line whose cut is a multi-charge dark vortex as described in Sec. 2.3.3. It would seem then (although in no way proven) that no high-charge single vortex rings exist. The multi-ring solutions are co-moving but suffer from eventual instabilities resulting in multiple dynamical effects such as Kelvin mode perturbations and vortex ring destruction and nucleation.

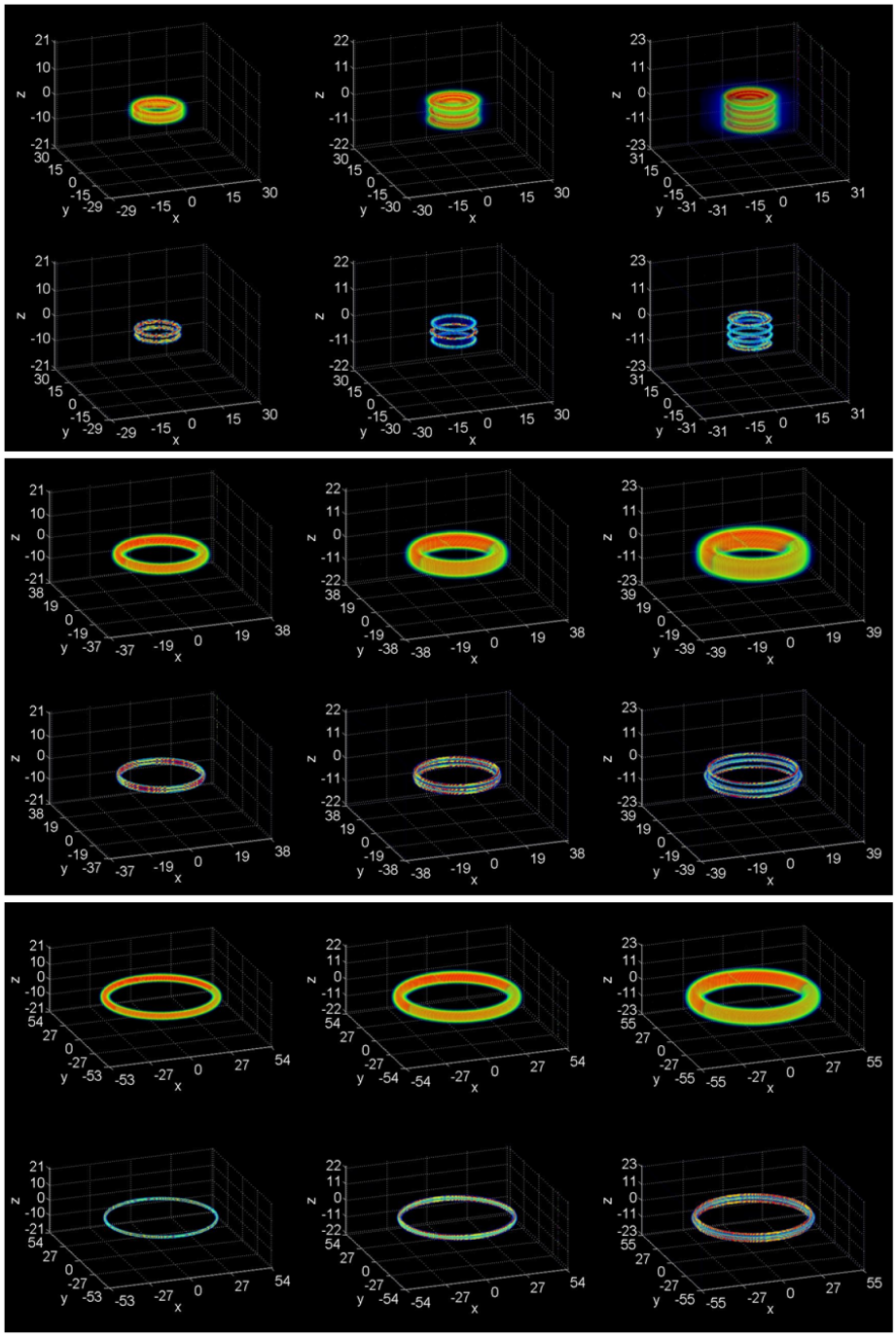


Figure 9.12. Examples of numerically optimized dark vortex ring solutions with the initial seed as a multi-charge vortex ring ansatz. Top to bottom sub figures are radii $d = 8, 16, 32$. In each sub figure, left to right are charges $m = 2, 3, 4$. For each combination of charge and radius, both the modulus-squared of Ψ and the modulus of the vorticity is shown. It is seen that in each case the optimization produces a multi-ring solution with the separation between rings smaller as d gets larger. The NLSE parameters are the same as in Fig. 9.5 and the spatial step size is $h = 2/3$.

We note that all of the co-moving multi-ring solutions obtained are quite interesting since, as we will see in Sec. 10.2.1, two vortex rings of charge $m = 1$ will rotate around each other in a leapfrog manner analogous to the similar behavior in classical vortex rings in fluids [136]. The fact that the rings in the present solutions can maintain their orientation for extended time-scales is interesting. A similar phenomenon was studied in Ref. [43] where steady-state parallel vortex rings were numerically realized in a trapped BEC setting. The rings there were found to be unstable over long times, similar to the co-moving multi-ring solutions found in this section.

CHAPTER 10

DYNAMICS OF MULTIPLE VORTEX RINGS

We now examine the dynamics of multiple interacting and colliding unitary-charged dark vortex rings. Our focus is on the interactions between two vortex rings, but the dynamics of larger numbers of vortex rings is explored as well.

10.1 SCATTERING OF TWO OPPOSITE-CHARGE DARK VORTEX RINGS

A study of the scattering of two unitary-charged ($|m| = 1$) dark vortex rings was performed in Ref. [38], where the authors focused on head-on collisions of the rings at different angled orientations. The results were discussed qualitatively without a quantitative analysis. In this section, a quantitative analysis of the scattering of two dark vortex rings is performed, focusing exclusively on offset co-planar collisions (not studied in Ref. [38])

If the planar offset of the vortex rings is set to zero, an axisymmetric head-on collision like that shown in Ref. [38] results. In such a case, the rings expand and annihilate each other's topological defect resulting in an axisymmetric decaying rarefaction pulse. An example of such a case is shown in Fig. 10.1 where two dark vortex rings of radius d (one with charge $m = 1$ and one with $m = -1$) are positioned a distance of $6d$ apart from each other in the z-direction and allowed to collide. The results shown in Fig. 10.1 should be contrasted to the qualitatively similar scenario of four two-dimensional dark vortices (two with $m = 1$ and two with $m = -1$), where the behavior of the vortices resembles the two-dimensional axisymmetric cut of the vortex rings. The resulting dynamics is shown in Fig. 10.2. We see that a major difference between the dynamics of the two-dimensional case of Fig. 10.2 versus the vortex ring case

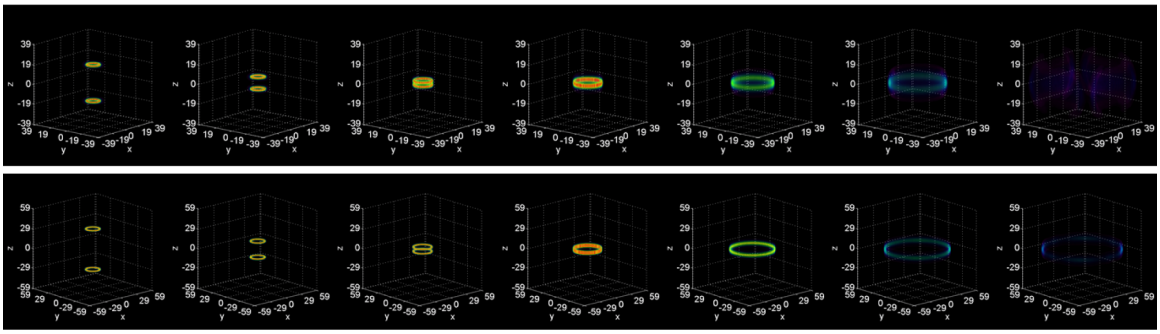


Figure 10.1. Examples of head-on collisions between dark vortex rings of charge $m = 1$ and $m = -1$. Top: Snapshots of rings with radius $d = 6$ at times $t = 0, 22, 33, 39, 45, 51, 59, 68$. Bottom: Snapshots of rings with radius $d = 10$ at times $t = 0, 47, 78, 91, 104, 116, 124$. The NLSE parameters in both simulations are $a = 1$, $s = -1$, and $\Omega = -1$, while the numerical parameters are $h = 0.8$ and $\mathrm{k} = 0.065$.

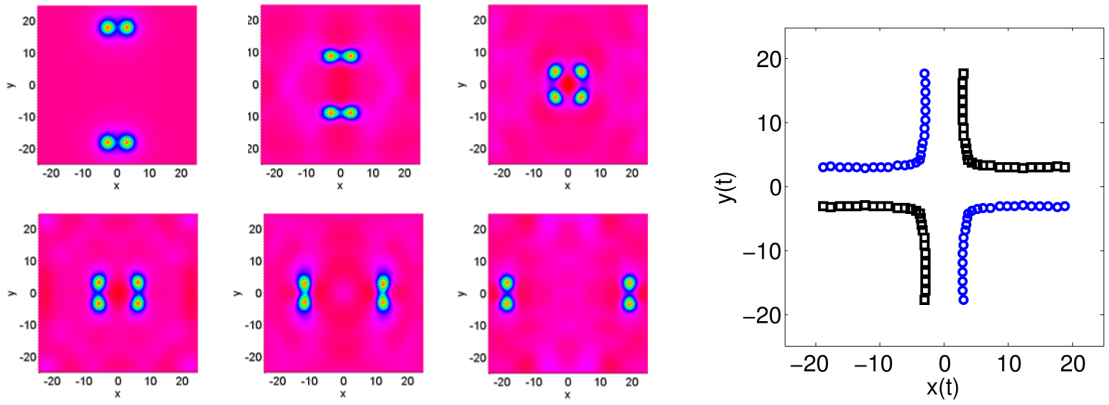


Figure 10.2. Four two-dimensional dark vortices (two of charge $m = 1$ and two of charge $m = -1$) in an analogous orientation of the collision of two dark vortex rings of radius $d = 6$ (shown in Fig. 10.1). Left: Snapshots of the vortices at times $t = 0, 26, 47, 57, 77, 98$. Right: Trajectories of the vortices (the (blue) dots represent the path of the vortices of charge $m = -1$, while the (black) squares are for those of charge $m = 1$). The NLSE parameters are the same as in Fig. 10.1, and the numerical parameters are $h = 1/3$ and $\mathrm{k} = 0.026$.

of Fig. 10.1 is that in the two-dimensional case, the vortices do not annihilate each other but instead avoid each other resulting in the vortices traveling in a perpendicular trajectory.

When the colliding vortex rings are offset from each other in the planar direction, scattering can result. To see the relationship between offset distance and scattering angle, the vortex rings are positioned in the y-direction with a planar offset distance defined as q. The scattering angle is defined as the angle away from vertical so that if the vortex rings travel in a straight trajectory, the angle is 0, while if they scatter at right-angles to each other, the angle is $\pi/2$. An example of a collision with offset $q/d = 1$ for vortex rings of radius $d = 6$ is shown in Fig 10.3. The two rings merge and then separate into two new vortex rings, each exhibiting a large quadrupole oscillation. To record the scattering angle of the vortex rings undergoing collision, a y-z cut of the computational grid is used to track the vortex ring's position as described in Chapter 9 (we note that in the case described below of the vortex rings resulting in an angled rarefaction pulse, the pulse can still be tracked as its vorticity is still greater than the surrounding background). The two-dimensional tracking can be used because, unlike the angled collisions of Ref. [38], the orientation in the x-y plane of the vortex rings undergoing planar offset collisions remains the same. To test the relationship between offset and scattering angle, a large number of simulations were run with an offset ranging from $q/d = 0$ to $q/d = 3$ in increments of 0.05 for vortex rings of radius $d = 6, 8, 10$. The results are shown in Fig. 10.4. It is clear from the results that there are three distinct sections of the scattering angle versus the offset, each representing three different topological outcomes. From offset $q/d = 0$ to $q/d = 0.5$, the two rings annihilate each other's topological defect sending out a rarefaction pulse similar to the case of the head-on collisions shown in Fig. 10.1 above, with two major differences. One, the rings rotate causing the collisions to occur at different angles depending on the offset, and two, the resulting rarefaction pulse is not radially symmetric in strength, but rather is more concentrated in the scattering direction. When the offset is greater than $q/d = 1.5$, the two rings undergo merging and

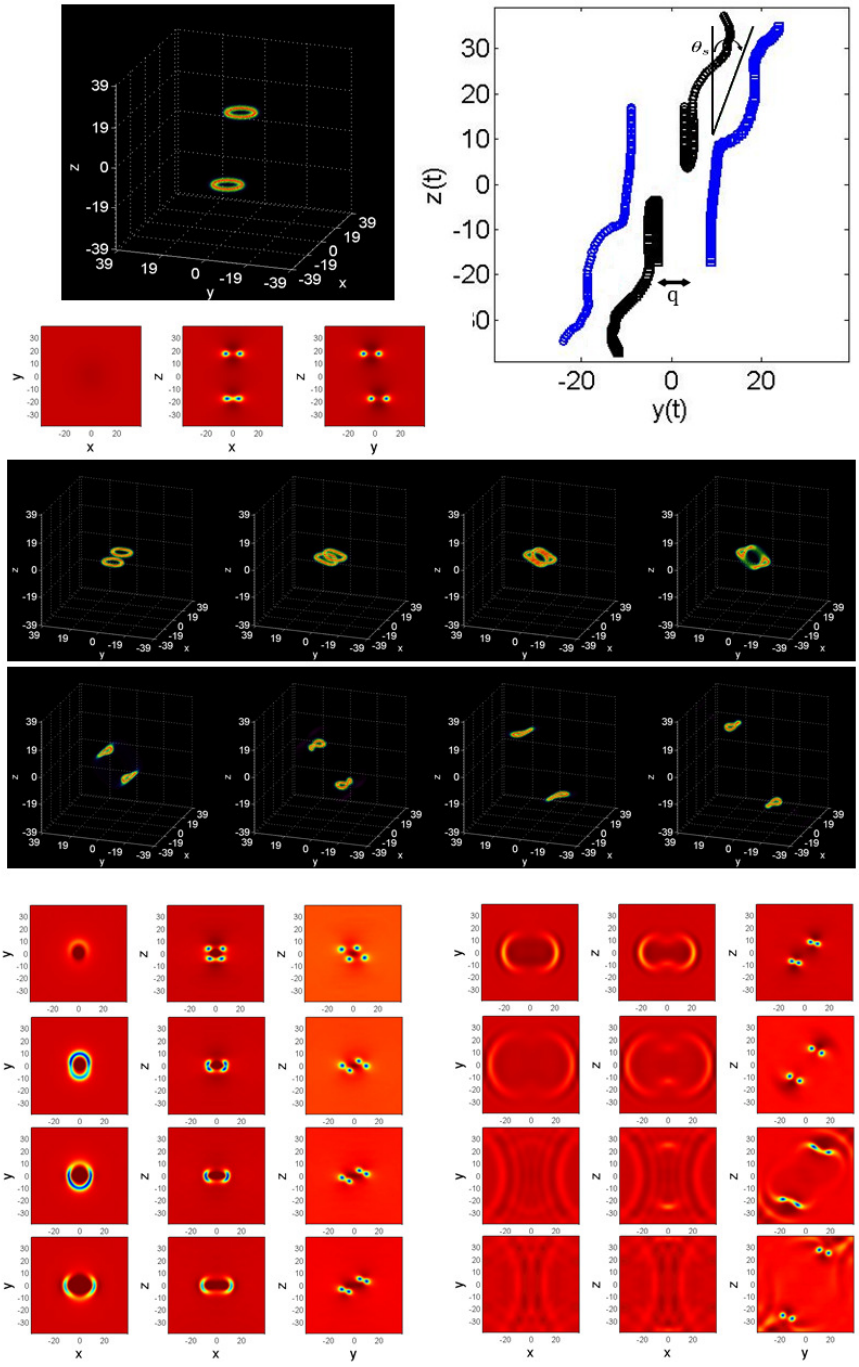


Figure 10.3. Collision of two dark vortex rings of radius $d = 6$ offset in the y-direction by $q/d = 1$. Top left: Volumetric rendering and two-dimensional cuts of the modulus-square of the initial condition. Top right: Trajectories of the two-dimensional cut of the scattering vortex rings showing the definition of the offset q and scattering angle θ_s. Bottom: Volumetric and two-dimensional cut snapshots of the scattering rings at times $t = 27, 33, 36, 39, 46, 54, 68, 77$. The NLSE and numerical parameters are the same as in Fig. 10.1.

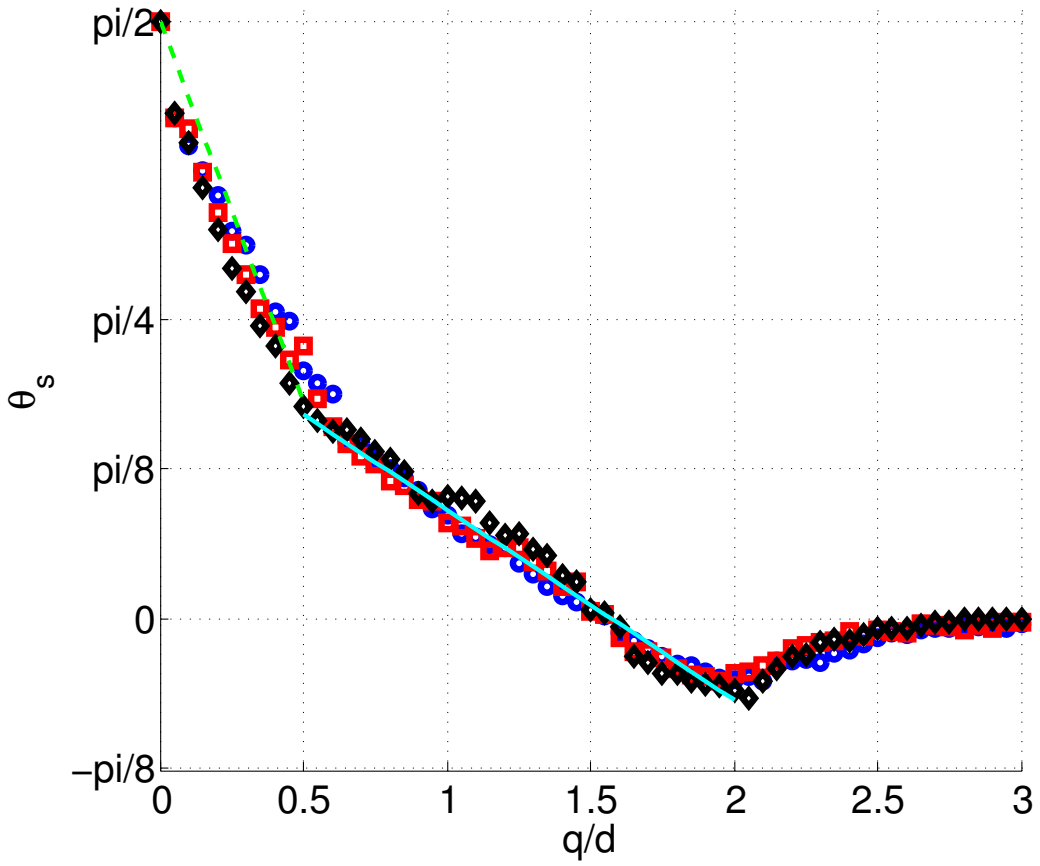


Figure 10.4. Scattering angle versus the offset distance q over the radius d for two colliding planar unitary-charge dark vortex rings. The (blue) circles are the results for vortex rings of radius $d = 6$, the (red) squares for $d = 8$ and the (black) diamonds for $d = 10$. The (green) dashed line is the approximation of Eq. (10.1) and the (cyan) solid line is the approximation of Eq. (10.2). The NLSE and numerical parameters used in the simulations are the same as in Fig. 10.1.

re-separating into quadrupole oscillatory rings as in the example shown in Fig 10.3. When the offset is set to $q/d = 1.5$ or larger, the scattering angle becomes negative, but the merging mechanism remains the same. This continues up to an offset of $q/d = 2$, where the two rings no longer merge and separate into new rings, but are merely perturbed by each other in a fly-by scenario causing their trajectories to travel at a negative angle from incidence. As the offset increases, this interaction decreases until approximately $q/d = 3$ where the rings ignore each other's presence and travel unperturbed. In Fig. 10.5 we display examples of the qualitatively distinct areas of Fig. 10.4.

Noting that the relationship between relative offset and scattering angle in Fig. 10.4 is linear in the first two qualitatively distinct areas, simple analytical approximations of the relationship can be formulated. For the annihilating rings ($q/d = 0$ to $q/d = 0.5$), the angle of the rarefaction pulse can be approximated by

$$\theta_s \approx -2\,\frac{q}{d} + \frac{\pi}{2}, \tag{10.1}$$

while for the scattering rings ($q/d = 0.5$ to $q/d = 2$), the angle of the resulting rings is approximated as

$$\theta_s \approx -\frac{1}{2}\,\frac{q}{d} + \frac{\pi}{4}. \tag{10.2}$$

Both approximations are shown with the simulation data in Fig. 10.4.

10.2 Dynamics of Two Equal-Charge Dark Vortex Rings

10.2.1 Leapfrogging Parallel Vortex Rings

To gain an idea of the interaction between two unitary-charged dark vortex rings of equal charge ($m_1 = m_2 = 1$) aligned along the z axis traveling together, we first show an example of a two-dimensional vortex scenario which is a qualitative analog of the axisymmetric cut of the two vortex rings. In Fig. 10.6, four dark vortices are set up similar

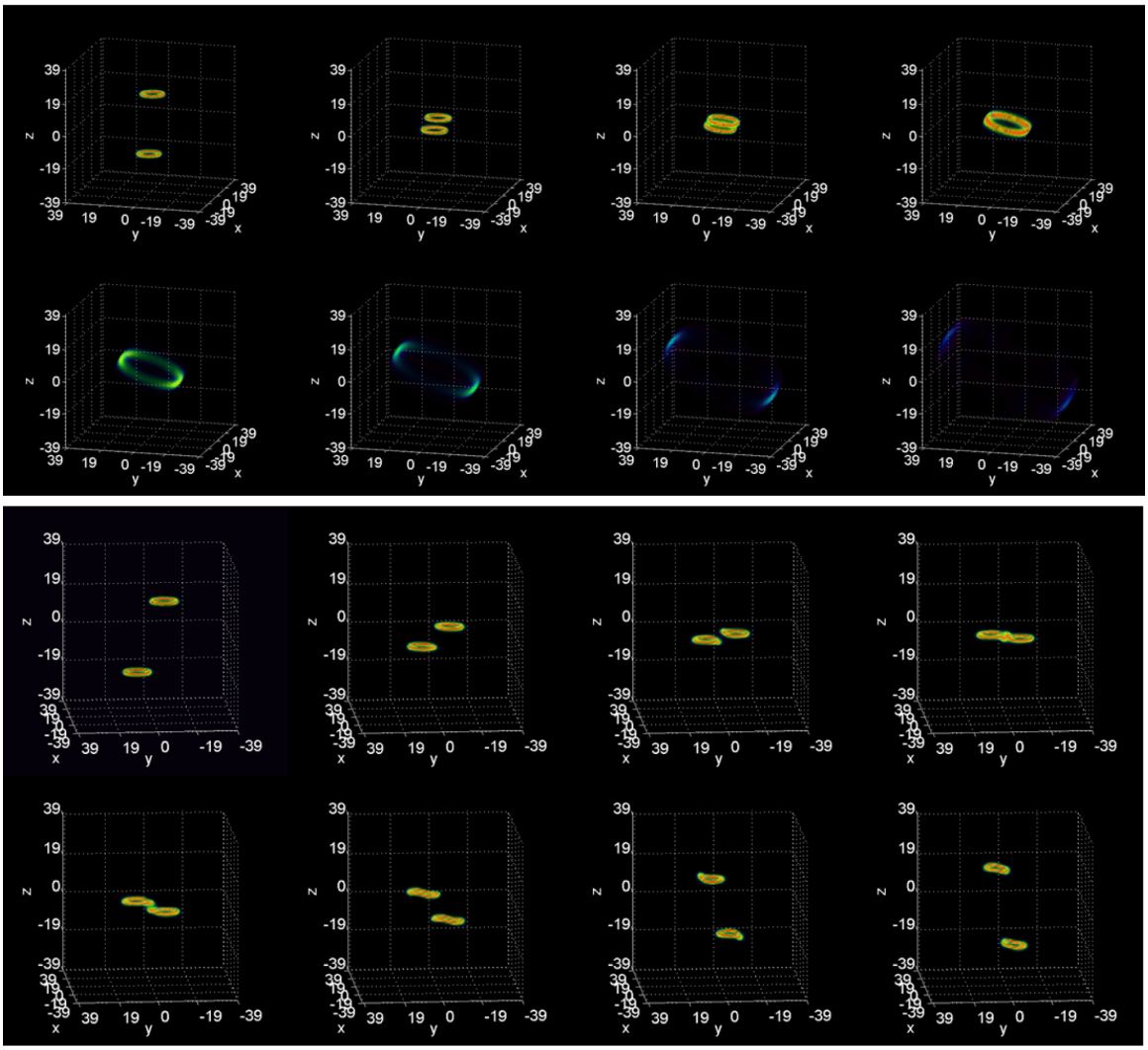


Figure 10.5. Qualitatively distinct scattering scenarios for two colliding offset planar dark vortex rings. Top: Snapshots of a rotated annihilation scenario (with initial offset of $q/d = 0.35$) at times $t = 0, 27, 33, 39, 45, 51, 58, 63$. Bottom: Snapshots of a non-merging interactive flyby (with initial offset of $q/d = 2.2$) at times $t = 0, 23, 29, 33, 36, 44, 58, 67$. The scenario of a merging and scattering vortex ring collision was already shown in Fig. 10.3. The NLSE and numerical parameters are the same as in Fig. 10.1.

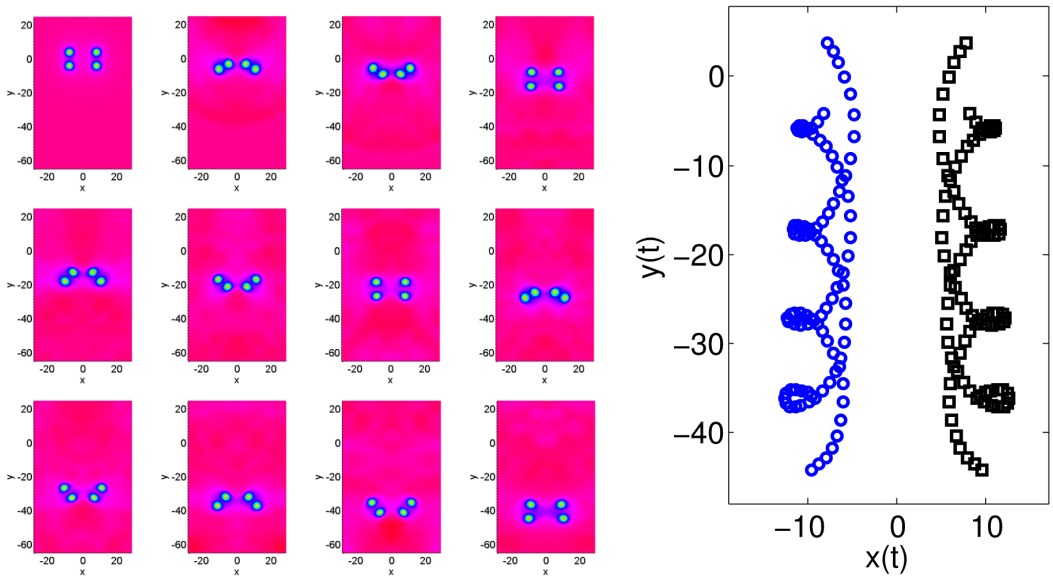


Figure 10.6. Simulation of four two-dimensional dark vortices arranged to be an analog to an axisymmetric cut of two equal-charged vortex rings of radius $d = 10$ separated in the z-direction by a distance $q = 5$. Left: Modulus-squared of the solution at times $t = 0, 19, 29, 49, 53, 79, 86, 115, 131, 159, 179, 196$ (left-to-right top-to-bottom). Right: Traced trajectories of the vortices. The description of the plot, as well as the NLSE and numerical parameters are the same as in Fig. 10.2.

to those in Fig. 10.2, where each vortex pair is separated by a distance of 5 with a distance of 20 between the pairs, and simulated to time $t = 200$. The resulting dynamics are shown in Fig. 10.6. The pairs of opposite-charged vortices rotate about each other, but the interactions of the equal-charge pairs remain, causing the four vortices to travel in the y-direction as well. We therefore expect that the vortex rings in the analogous orientation will leapfrog around each other as they travel in the z-direction, similar to the leapfrogging dynamics of vortex rings in fluids [136].

In Fig. 10.7, we show an example of two equal-charge vortex rings of radius $d = 10$ separated by a distance of $q = 5$ in the z-direction undergoing leapfrogging dynamics. Through numerical tests, we note that the frequency of the leapfrogging rings changes depending on the z-distance between the rings. However, when q is small (around two or three times the healing length), or when d is small compared to q, the resulting dynamics are quite different. Instead of leapfrogging, the rings start to travel together and

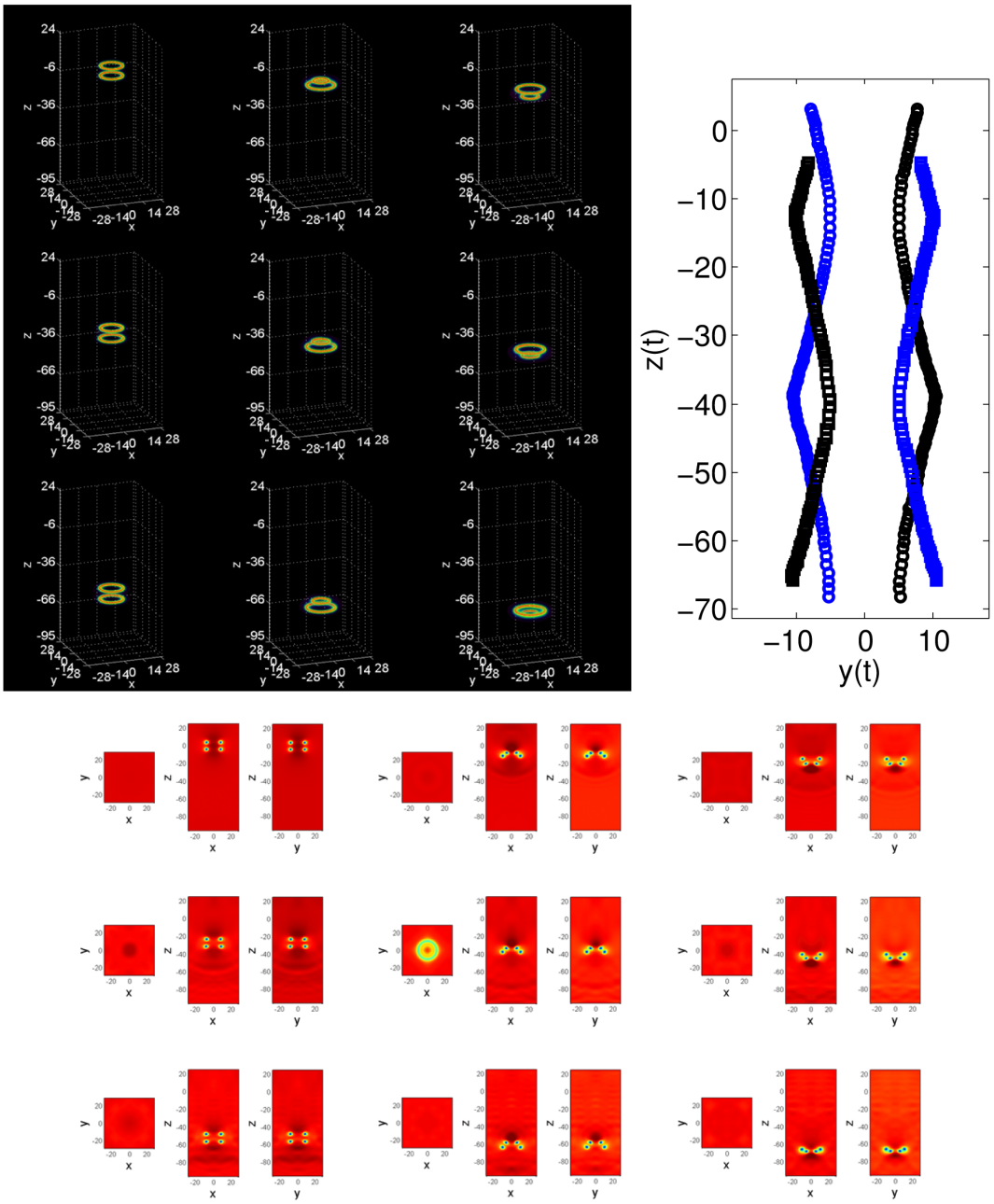


Figure 10.7. Simulation of two dark vortex rings of radius $d = 10$ separated in the z-direction by a distance $q = 5$. Top Left: Modulus-squared of the solution at times $t = 0, 16, 27, 44, 58, 68, 85, 101, 110$ (left-to-right top-to-bottom). Bottom: Two-dimensional cuts of the solution (the times displayed are the same as above). Top Right: Traced trajectories of the vortex rings. The description of the plot, as well as the NLSE parameters are the same as in Fig. 10.1. The numerical parameters are $h = 2/3$ and $\mathrm{k} = 0.045$.

then a new vortex ring is nucleated in the center of the two rings which then collides with the bottom ring either sending out a rarefaction pulse or splitting the rings into multiple vortex rings traveling in near-transverse directions. The dynamics are the same as those described in Sec. 9.3 for the break-up of the co-moving multi-ring ensemble solutions. As mentioned there, the detailed study of such dynamics are beyond the current scope and therefore, in the current section, we focus on vortex rings separated with large enough values of q to result in leapfrogging dynamics.

Two relevant quantities to study in the case of leapfrogging rings are their average velocity and frequency of oscillation. Using the same tracking procedure described in Sec. 10.1, the velocity and frequency of the leapfrogging rings can be extracted from the simulations. In Fig. 10.8, we show the results of tracking the average transverse velocity of the rings. The results are shown for varying the z separation distance q for a few values of d, and by varying d for a few values of q. In each case, we compare the results to the predicted values of the transverse velocity given by Eq. (2.39) for both a charge $m = 1$ and $m = 2$ vortex ring. We see that the average velocity of the leapfrogging rings is much closer to the predicted velocity of a charge $m = 2$ vortex ring than that of a charge $m = 1$. This is understandable since the phase structure of the leapfrogging rings (far from the rings) is almost the same as would be for a charge $m = 2$ vortex ring. It is also noticeable that the velocity seems to only be weakly dependent on the separation distance q (for the values tested). We note that the non-uniform behavior in the results of vortex rings of radius $d = 6$ for low values of q shown in the bottom left panel of Fig. 10.8 is most likely due to the combination of having a low value of d and q, in which case the interactions starts to approach those of the non-leapfrogging dynamics mentioned above. It appears that the smaller the value of q is, the closer the velocity gets to that predicted for a charge $m = 2$ vortex ring. This is easily understood as in the limiting case of $q \to 0$, the resulting solution would indeed resemble a vortex ring of charge $m = 2$ (however, in practice, as discussed above and in Sec. 9.3, when q gets small, non-leapfrogging dynamics occur).

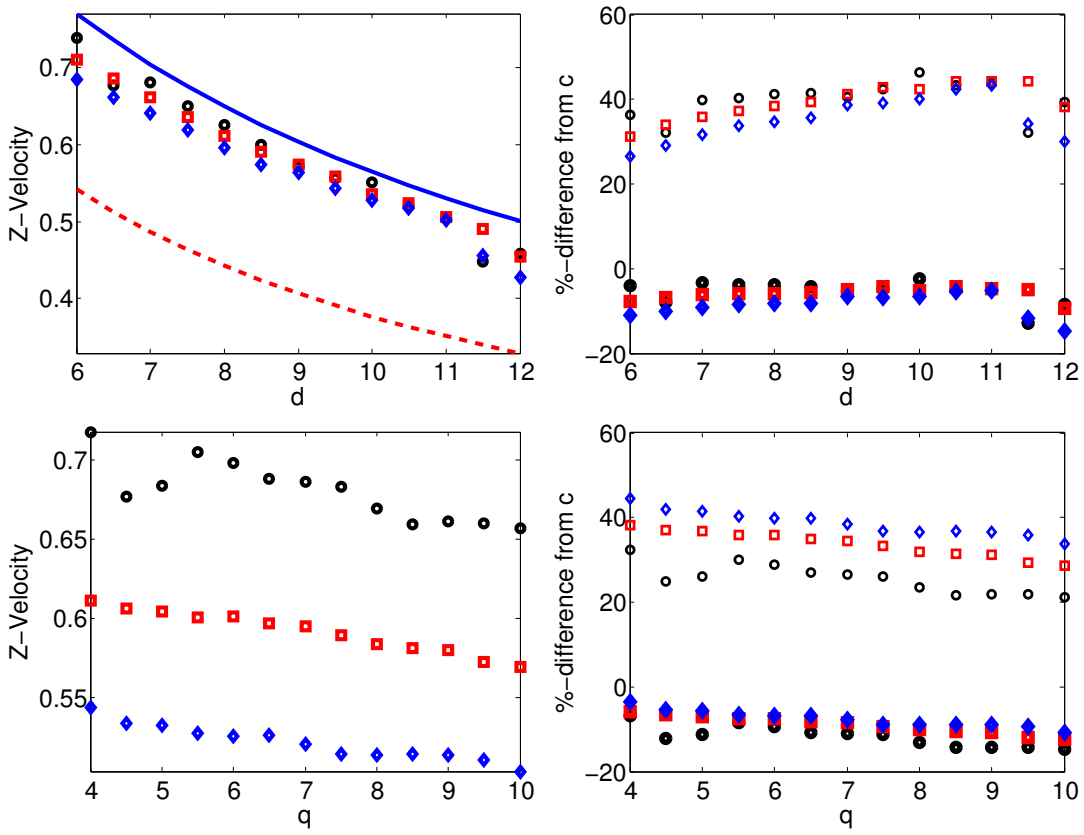


Figure 10.8. Average velocity versus separation distance (q) and radius (d) of two leapfrogging vortex rings. Top row: Velocity versus radius (left) and percent differences (right) from Eq. (2.39). The solid (blue) and dashed (red) lines are the predicted velocities of Eq. (2.39) for a vortex ring of charge $m = 1$ and $m = 2$ respectively. The (black) dots, (red) squares, and (blue) diamonds are results with separation $q = 4, 6, 8$ respectively. The smaller markers in the right plot are the percent differences to Eq. (2.39) with charge $m = 1$, while the larger markers are for $m = 2$. Bottom row: Velocity versus separation distance q (left) and percent differences from Eq. (2.39) (right) for leapfrogging vortex rings of radius $d = 6$ (dots), $d = 8$ (squares), and $d = 10$ (diamonds). The NLSE and numerical parameters are the same as in Fig. 10.7.

Next, we compare the loop-frequency versus the separation distance q for a few values of d, as well as the frequency versus d for a few values of q. The frequencies are found by tracking the y position of the vortex ring cuts and extracting the period of oscillation. The results are shown in Fig. 10.9. The frequency appears to be nearly

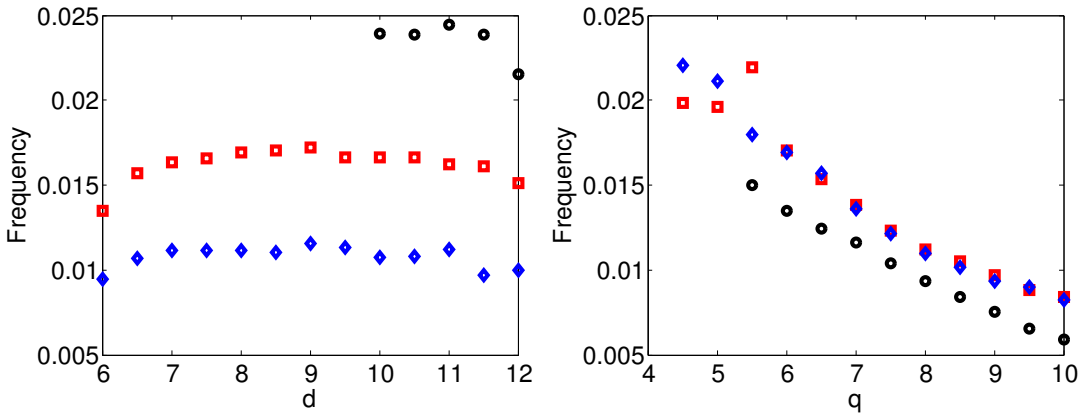


Figure 10.9. Frequency versus separation distance (q) and radius (d) of leapfrogging vortex rings. Left: Frequency versus radius for $q = 4$ (dots), $q = 6$ (squares), and $q = 8$ (diamonds). The missing $q = 4$ results are due to the tracking code failing to resolve the separate rings adequately during the simulation. Right: Frequency versus separation distance for radius $d = 6$ (dots), $d = 8$ (squares), and $d = 10$ (diamonds). The NLSE and numerical parameters are the same as those in Fig. 10.8.

independent of the radius, as long as the radius is large compared to q. The dependence of the frequency on q is quite pronounced, but does not seem to lead to any simplistic linear approximations like we found for the scattering results of Sec. 10.1.

As a last test of the leapfrogging vortex rings, we qualitatively investigate their stability by integrating the solutions for extended time periods. In order to be able to integrate the solution and not require an overly-large compute grid, we once again take advantage of the MSD boundary condition of Chapter 4 and place the leapfrogging rings amidst a co-moving back-flow with a velocity set by Eq. (2.39) with $m = 2$. An example of the simulations performed is shown in Fig. 10.10. The leapfrogging rings stay near the center of the grid for a long periods of time (here, over 650 time units) with a near-constant frequency. Then, the frequency starts to decrease, and the rings start to rise

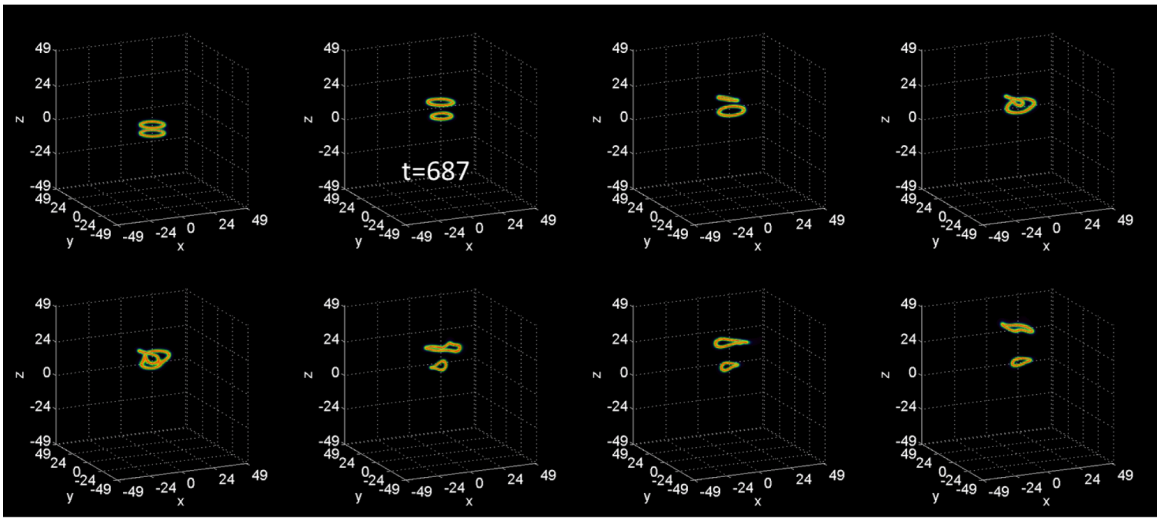


Figure 10.10. Modulus-squared of a long-time simulation of leapfrogging vortex rings of radius $d = 8$ initially separated by a z-distance of $q = 6$ in a co-moving back-flow with a velocity given by Eq. (2.39) with $m = 2$. The rings are shown at times (left-to-right, top-to-bottom) $t = 0, 687, 715, 731, 739, 755, 771, 811$. The NLSE and numerical parameters are the same as in Fig. 10.8.

in the computational box. After a while, one of the rings starts to rotate and then merges with the other ring creating two new scattered perturbed rings (just like the case of the break-up of the co-moving double-ring solutions of Fig. 9.11). After running several simulations, it is found that the larger the computational box, the longer the leapfrogging rings last before merging. Therefore, it seems that numerical errors are perturbing the rings which then causes them to eventually merge. This indicates that the leapfrogging ring solution is (weakly) unstable.

10.2.2 Merging Co-Planar Vortex Rings

As was demonstrated in the numerical simulations of Ref. [22], two equal-charge co-planar vortex rings will attract each other and eventually merge forming a single ring with an extremely large mode-two perturbation. An example of such a merger of two rings separated by a distance q (where q is defined as the distance between the closest edge cores of the rings) is shown in Fig. 10.11. The dynamics can be qualitatively explained as the two vortex ring edges wanting to travel in the opposite direction due to the charge

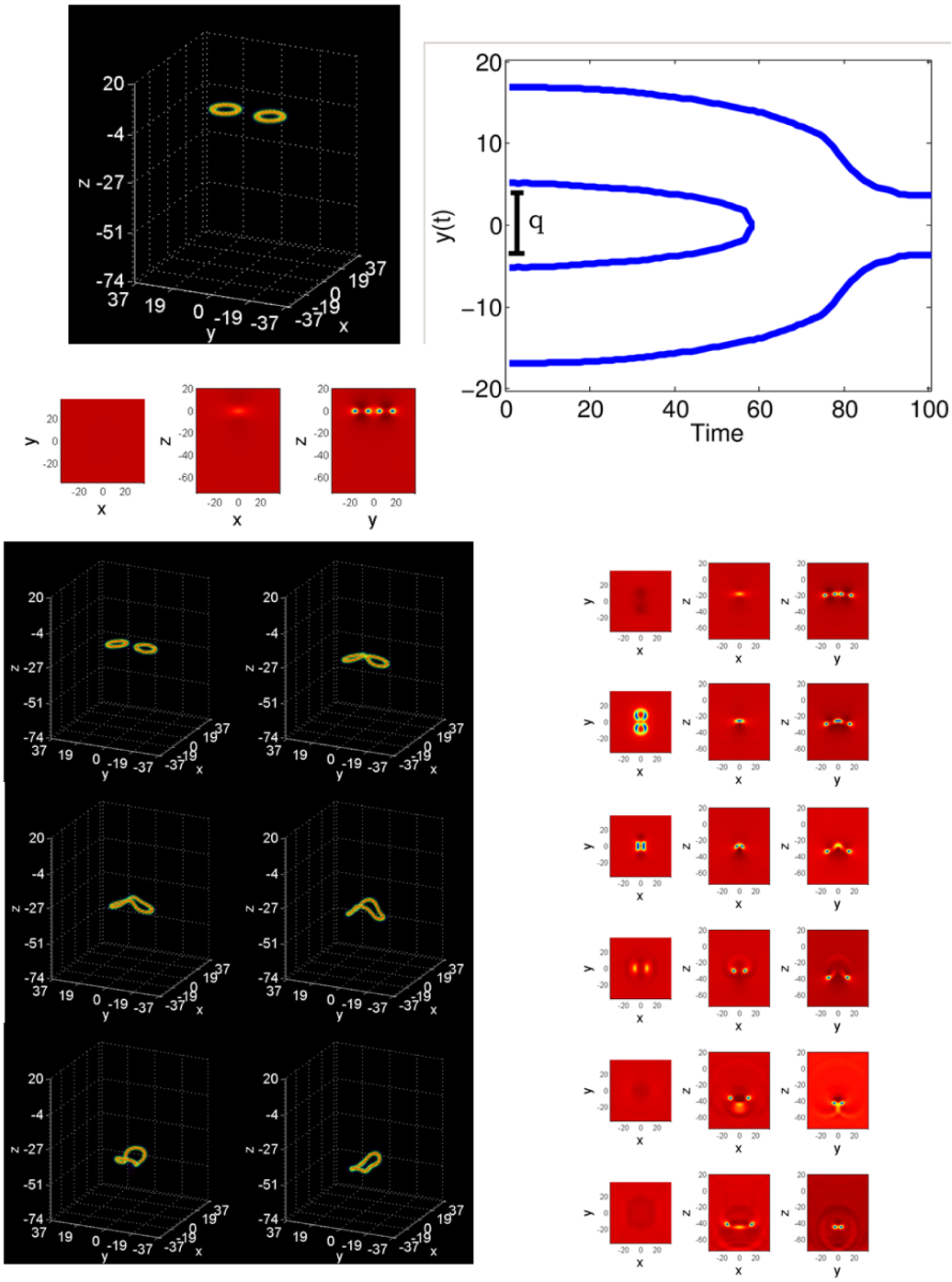


Figure 10.11. Simulation of two dark vortex rings of radius $d = 6$ separated from each other in the y-direction by a distance $q = 10$. Top Left: Modulus-squared of the initial condition. Top Right: Tracking of the y position of the vortex rings over time showing the definition of q. Bottom: Snapshots of the simulation at time $t = 0, 35, 53, 60, 69, 84, 96$. The NLSE and numerical parameters are the same as in Fig. 10.1.

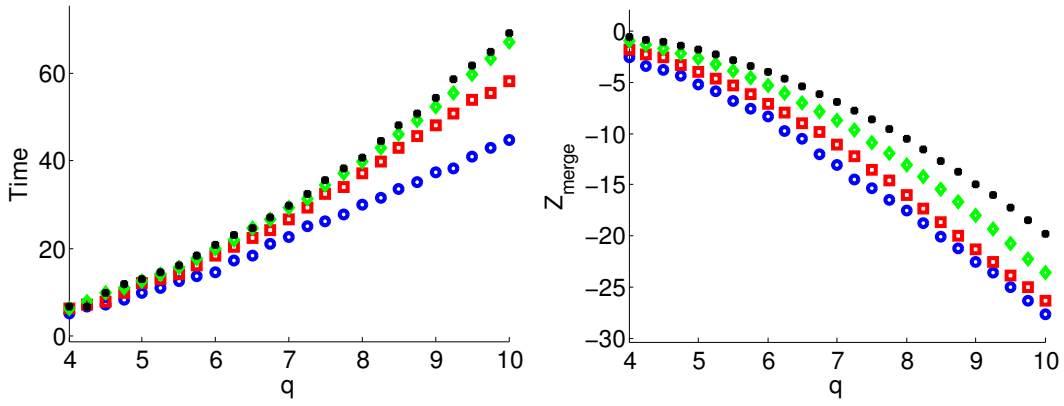


Figure 10.12. Time (left) and z-position (right) as a function of separation distance q of the merger of two co-planar vortex rings. The (blue) circles, (red) squares, (green) diamonds, and asterisks represent the results for vortex rings of radius $d = 4, 6, 8, 10$ respectively. The NLSE and numerical parameters are the same as in Fig. 10.1.

configuration along their aligned two-dimensional cut, and so cause the vortex rings to bend. This bending then changes the trajectory of the vortex rings as they approach each other until they merge. Therefore, the merge time will depend not only on the separation between the vortex rings, but also on the rings' radii (since the radius of a vortex ring determines its transverse velocity). In Fig. 10.12, we show the merging time and z-position of two co-planar vortex rings of equal charge whose center positions are separated from each other in the y-direction by a distance of $2\,d + q$ for rings with radius $d = 4, 6, 8, 10$. As expected, the time of merger is longer for larger rings due to their slower translational velocity, and correspondingly, their travel distance (in z) before the merge. However, as was the case in the leapfrog results of Sec. 10.2.1, no simple model approximation to the current results is apparent. This is understandable as the velocity of the rings depends on d in a nontrivial way [see Eq. (2.39)], and, likewise, the 'pull' that the sides of the vortex rings 'feel' that causes their rotation.

10.3 Choreographed Multiple Interacting Vortex Rings

The number of possible interactions between multiple vortex rings is vast. Previous studies involving multiple vortex rings have typically focused on vortex rings in

random orientations exhibiting quantum ‘tangle’ (turbulence) [50]. In this section, we design some interesting interacting multiple vortex ring scenarios as an extension to the previous results of this chapter, as well as an opening for further research.

In Sec. 10.2.2, we saw that two equal charge co-planar vortex rings will attract each other and then merge. Our first choreographed multi-vortex ring example is setting four co-planar vortex rings symmetrically and allowing them to merge. The results are shown in Fig. 10.13. The rings merge and create a small vortex ring with opposite charge which then travels in the opposite direction leaving behind one large vortex ring with large Kelvin mode perturbations (in this case an octupolar mode).

Placing the four rings so that they are all at right angles facing each other yields the result shown in Fig. 10.14. The four rings travel directly to each other and merge to form two opposite charge vortex rings traveling away form each other exhibiting a large octupolar mode. The results of Figs. 10.13 and 10.14 imply that four merging vortex rings will produce two opposite charge vortex rings, where the radius of each produced ring is dependent on the angle of the four colliding rings.

If an additional vortex ring is placed in the center of the four merging rings shown in Fig. 10.13, different dynamics result as shown in Fig. 10.15. The rings merge into a flowering large ring which then recombines on itself forming the original five rings. However on the second recombination, the four ‘petals’ of the flowering ring break off below the center ring. The four lower rings merge and then shoot a ring back through the center ring. In other tests with a smaller grid, the secondary dynamics are observed initially, while for larger grids, the rings survive through two periods of recombination. Therefore the five-ring configuration is quite sensitive to perturbations (and to the symmetry of the initial positions of the rings).

Using the scattering results of Sec. 10.1, we can set up ‘billiard’ collisions. For example, two or more scattering vortex rings can be arranged so that the scattered rings collide and scatter themselves. In Fig. 10.16, we show an example of four pairs of off-set

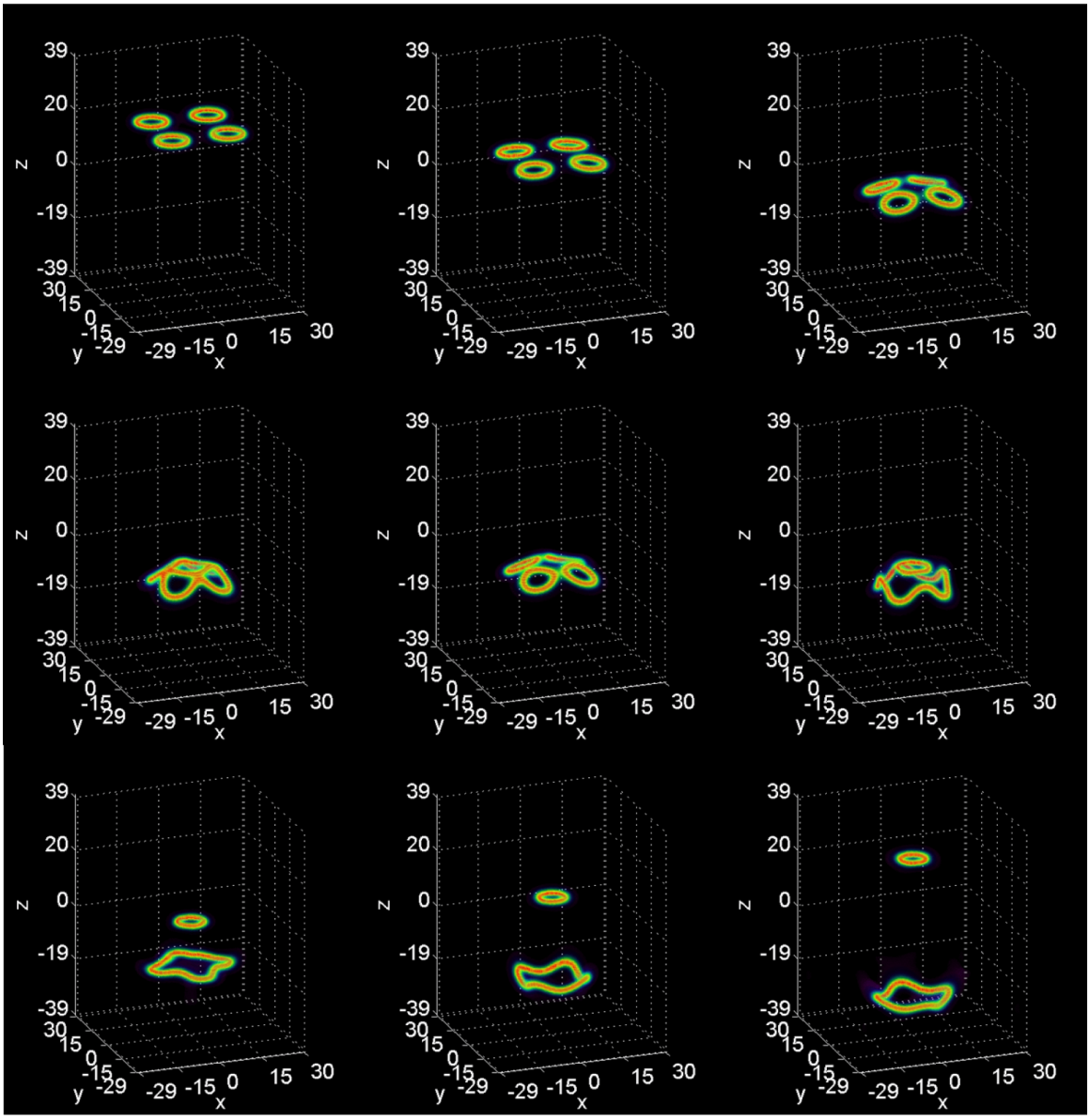


Figure 10.13. Merger of four co-planar dark vortex rings of radius $d = 5$ placed a distance of $10\sqrt{2}$ from the center of the configuration. The solution is shown at times $t = 0, 19, 43, 49, 55, 60, 74, 88, 110$ (left-to-right, top-to-bottom). The NLSE parameters are the same as in Fig. 10.1, while the numerical parameters are $h = 1/2$ and $\mathrm{k} = 0.034$.

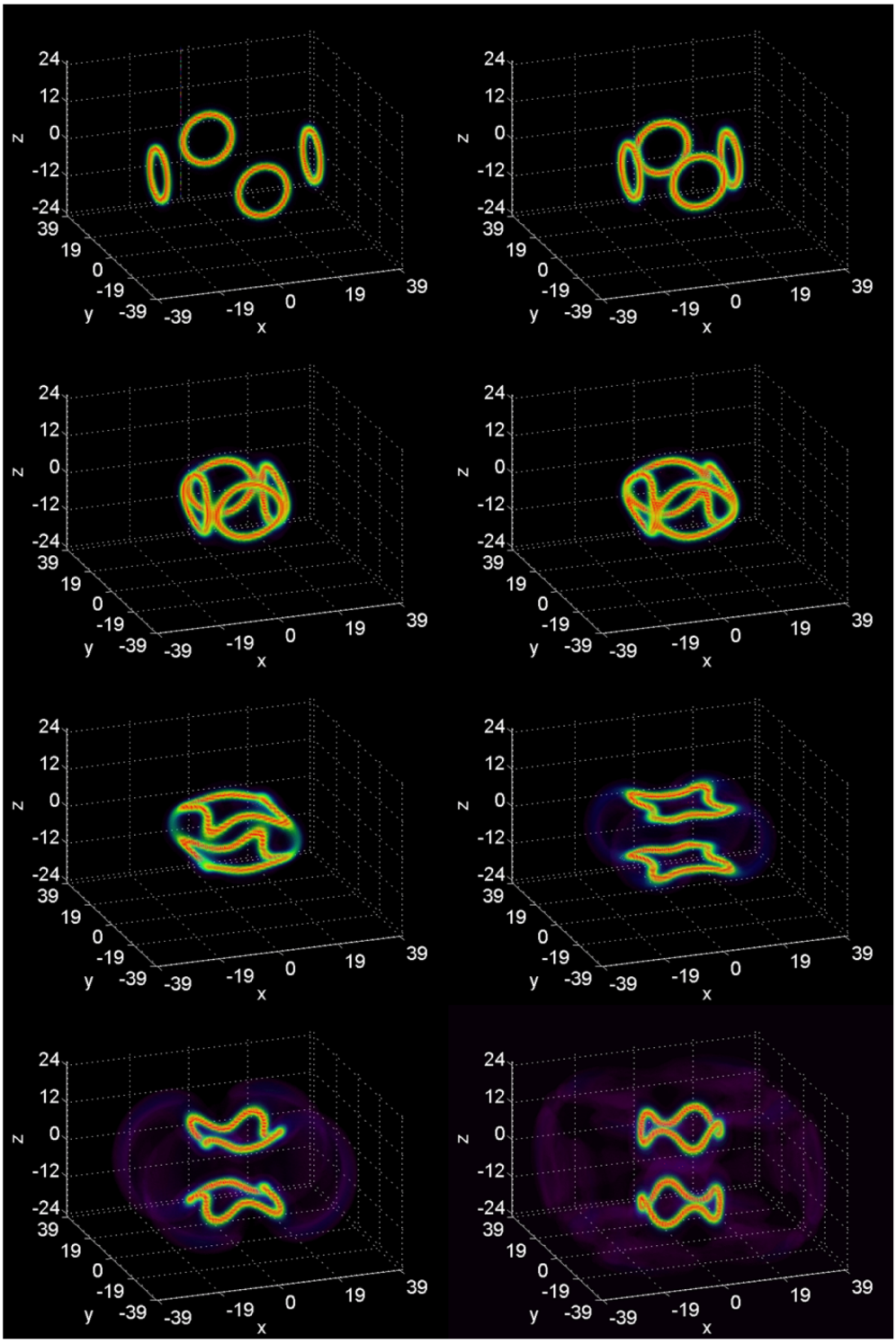


Figure 10.14. Merger of four co-planar dark vortex rings of radius $d = 8$ placed a distance of $25\sqrt{2}$ from the center of the configuration at right angles so the rings all face each other. The solution is shown at times $t = 0, 22, 37, 40, 43, 49, 55, 62$ (left-to-right, top-to-bottom). The NLSE parameters are the same as in Fig. 10.1, while the numerical parameters are $h = 2/3$ and $\mathrm{k} = 0.045$.

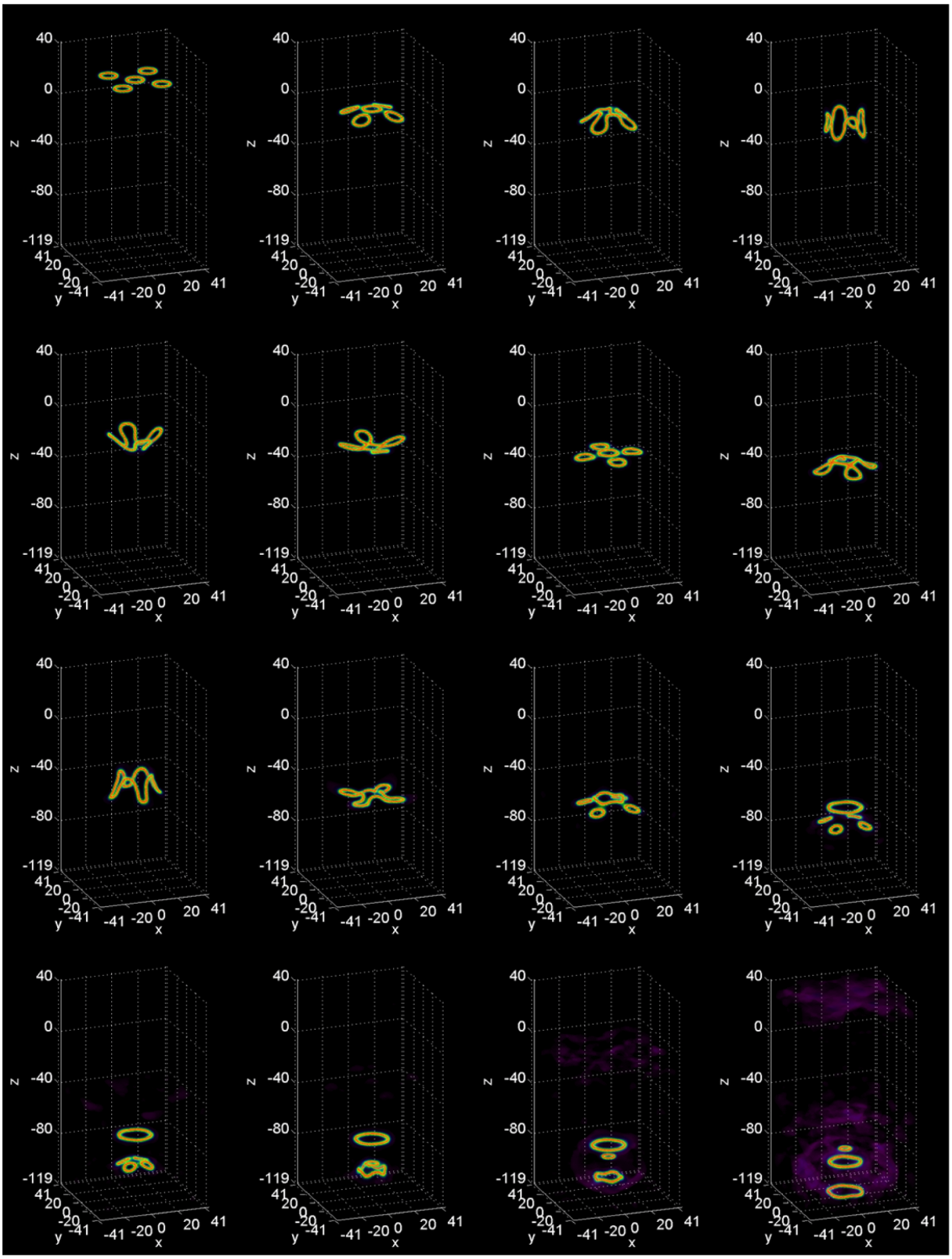


Figure 10.15. Merger of five co-planar dark vortex rings of radius $d = 6$ placed at a distance of $15\sqrt{2}$ between each ring and the center ring. The solution is shown at times $t = 0, 54, 68, 84, 100, 115, 134, 156, 173, 215, 230, 253, 286, 296, 310, 345$ (left-to-right, top-to-bottom). The NLSE parameters are the same as in Fig. 10.1, while the numerical parameters are $h = 3/4$ and $\mathrm{k} = 0.058$.

vortex rings arranged so that the scattered rings merge together into a four-ring merger like that of Fig. 10.13. The four scattered rings converge in the center as expected and merge, releasing a new vortex ring in the opposite direction, and leaving behind a single large ring. It is interesting to note that the merging of the four vortex rings produces the same dynamics as in Fig. 10.13 even though the rings each possesses a large quadrupole oscillation at the time of collision.

As a final configuration, we simulate two charge $m = 3$ co-moving triple-stacked vortex rings as described in Sec. 9.3 colliding into each other in both a head-on and an offset collision (with the offset distance equal to the ring radius). The resulting dynamics are shown in Fig. 10.17. In the head-on case, two single vortex rings traveling away from the center of the collision survive. In the offset collision, the scattered vortex rings get entangled, eventually causing multiple small rings to break off from the scattered group. However, the scattering angle of the overall entangled group seems to be similar to that exhibited in the two ring collisions of Sec. 10.1.

The number of configurations of possible multi-vortex ring interactions is endless and therefore we leave the simulation of further choreographed vortex ring interaction scenarios to the reader through the use of the NLSEmagic code package.

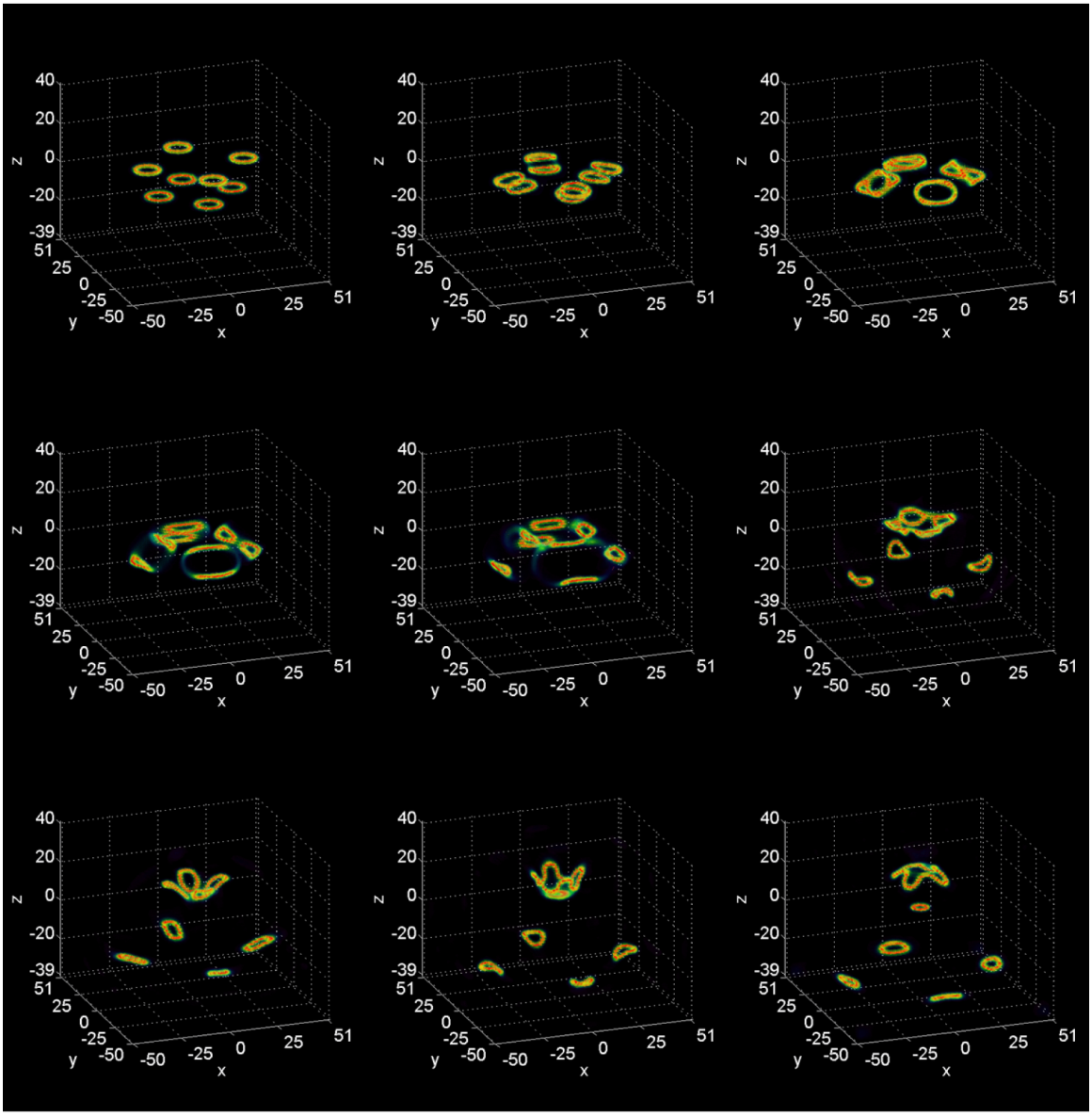


Figure 10.16. Eight scattering vortex rings of radius $d = 6$ resulting in a four-ring merger. The ring pairs are a distance of 15 apart from each other in the z-direction, and offset by $q/d = 1$. The top rings are arranged a distance of $25\sqrt{2}$ in the x–y plane from the center of the grid. The solution is shown (left-to-right, top-to-bottom) at times $t = 1, 13, 19, 23, 27, 38, 47, 55, 67$. The NLSE parameters are the same as in Fig. 10.1, while the numerical parameters are $h = 1$ and $\mathrm{k} = 0.1$.

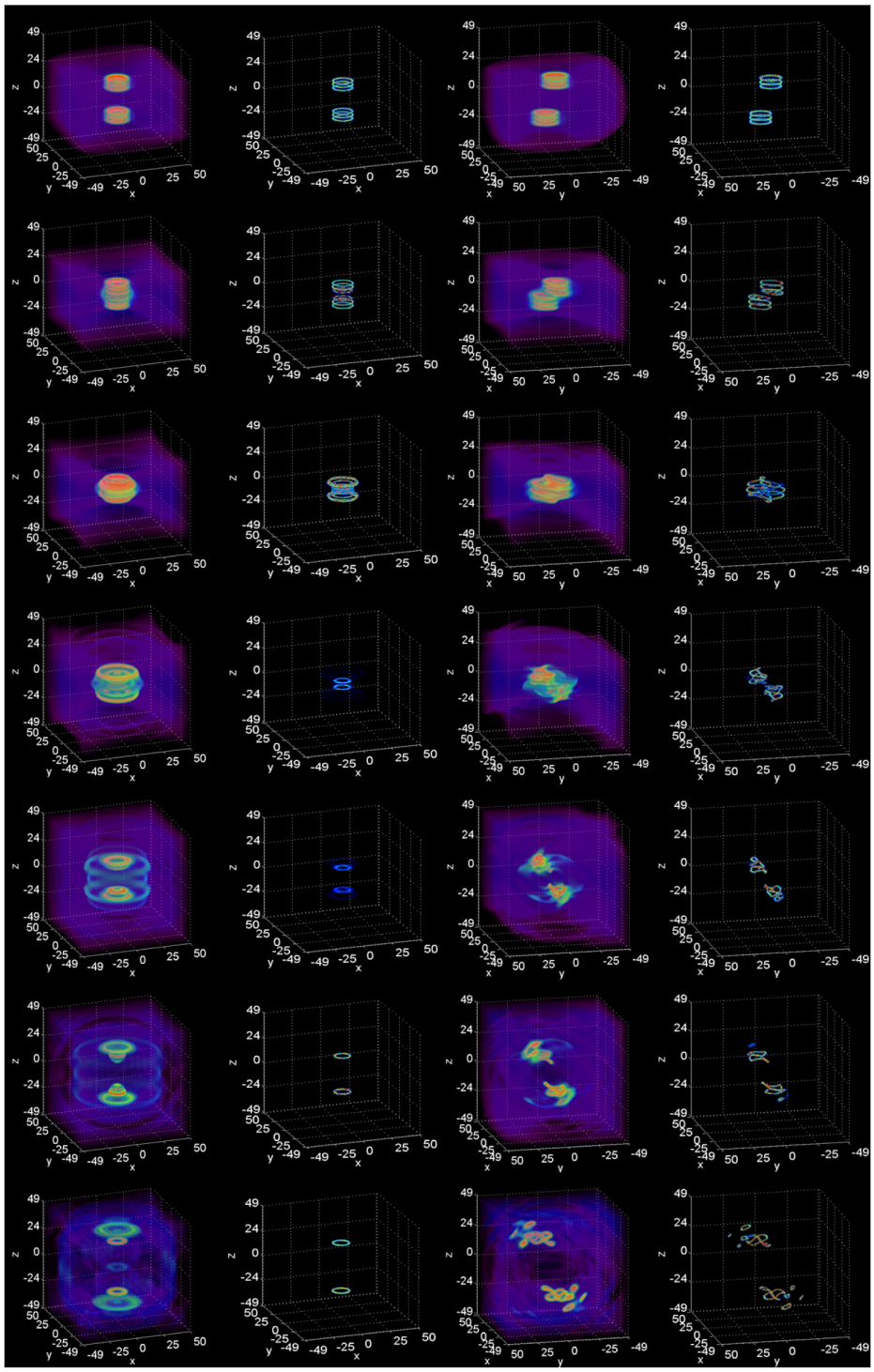


Figure 10.17. Two charge $m = 3$ co-moving vortex multi-rings (see Sec. 9.3) of radius $d = 8$ colliding. The ring ensembles are a distance of 30 apart from each other in the z-direction. The left two columns show the modulus-squared and vorticity of a head-on collision at times $t = 0, 11, 20, 27, 37, 47, 58$, while the right two columns show an offset collision with offset $q/d = 1$ at times $t = 0, 13, 20, 27, 36, 43, 57$. The NLSE and numerical parameters are the same as in Fig. 10.14.

CHAPTER 11

CONCLUSIONS

In this dissertation, we have designed and developed a redistributable GPU-accelerated MATLAB-based code package called NLSEmagic to integrate the nonlinear Schrödinger equation (NLSE) with high-order compact finite difference methods and used it to numerically study the dynamics of vortex rings in the three-dimensional NLSE including interactions, collisions, and scattering of multiple vortex rings. In this chapter, we summarize the results, mention publications arising from the work, address reproducibility of research, and mention ideas for further extensions.

11.1 SUMMARY OF RESULTS

The study of vortex rings in the NLSE conducted in this dissertation has led to several interesting results. We realized numerically-exact dark vortex ring solutions by numerical optimization and, through direct integration, confirmed previous analytical formulations of their transverse velocity as a function of their radius. Bright toroidal vortex rings (not previously reported in the literature) of multiple charges were also formulated, and their collapse into singularities observed. By attempting to optimize high-charged dark vortex rings, a seemingly previously unknown family of co-moving multi-ring ensemble solutions were found. These co-moving stacked-ring structures were found to be unstable over long periods of time. Quadrupole oscillations of single-charged dark vortex rings caused by a mode-two planar perturbation were studied, and their frequency and velocity was found as a function of the perturbation amplitude. This yielded simple linear approximations to these relationships which could be useful in designing experiments of vortex rings in Bose-Einstein condensates (BEC).

The interactions between two unitary-charged dark vortex rings were investigated yielding a variety of dynamics depending on their mutual orientation, and the sign of their charge. For opposite-charged vortex rings, offset co-planar collisions were studied. The results showed that the ensuing dynamics only depends on the ratio of their radius to the offset distance. Namely, for low separations, the rings collide at an angle, sending out a rarefaction pulse concentrated in the alignment direction of the two rings, while higher offset values cause the rings to merge and form two new scattered rings. In either case, the angle of the resulting solution is dependent on the offset distance. The results yielded simple approximations to the scattering angle as a function of relative offset for each of the qualitatively distinct scenarios.

Two orientations of two interacting dark vortex rings of equal sign charge were explored — two rings aligned along the z-axis, one on top of the other, and two rings set co-planar (side-by-side). The rings set one on top of the other resulted in leapfrogging dynamics where the frequency of the looping was found to be highly dependent on the separation of the rings in the z-direction, but with very little dependence on the rings' radii. The average velocity of the looping pair of rings was found to be much closer to the predicted velocity of a hypothetical charge-two vortex ring, than that of a charge one. When the two rings were placed side-by-side, the rings traveled together but were attracted to one another causing them to merge into one vortex ring with a quadrupole oscillation. The time and position of the merger was shown as a function of the separation distance of the rings. Simulations of multiple vortex rings together were performed resulting in a wide variety of dynamics including mergers, scattering, and the nucleation and annihilation of vortex rings.

The code package used to conduct the simulations of the vortex ring solutions uses a two-step high-order compact (2SHOC) scheme for the Laplacian operator. While the numerical analysis theory behind the fourth-order scheme is not novel, its specific formulation and use as a compact scheme has not been previously utilized. Its three main

advantages are that a) its compact nature makes it easier to actualize in parallel code environments, b) its independent two-step procedure allows it to be used in fully explicit time-stepping schemes, and c) unlike most other high-order compact schemes, it is not specific to any one particular governing equation.

To integrate the vortex ring solutions in an infinite background density environment (for example, simulating a homogeneous BEC), a new modulus-squared Dirichlet boundary (MSD) condition was developed which keeps the modulus-squared of the solution constant at the boundaries. Through several numerical tests, the MSD boundary condition was shown to be very useful in the present context. The MSD boundary condition has the advantage of being a general method applicable to a host of governing equations, as well as being very easy to implement.

The overall numerical scheme used in integrating the NLSE was made by combining the 2SHOC scheme for the Laplacian, the MSD boundary conditions, and a fourth-order Runge-Kutta time-stepping scheme, yielding a fully explicit numerical method. In order to make efficient use of the method, a linearized stability analysis was performed in order to be able to use (given a desired spatial step-size) as large of a time step-size as possible. The analysis involved using the Gershgorin circle theorem to find bounds on the maximum eigenvalues of the matrix form of the linearized schemes. The results were tabulated into an easy-to-reference two-page appendix, and are automatically computed in the NLSEmagic code package.

The numerical methods were then implemented into the NLSEmagic distributable code package. The codes use a combination of MATLAB, C, and NVIDIA graphics processing unit (GPU) compute unified device architecture (CUDA) codes to integrate the NLSE. The MATLAB driver scripts are used to set up initial conditions, visualizations, data analysis, etc., while the main computations are performed on compiled C codes with a MEX input/output interface, allowing them to be called from the MATLAB scripts. The C integrator codes were shown to run much faster than using MATLAB script codes. In

order to be able to use the code for large three-dimensional simulations efficiently, a set of GPU-accelerated CUDA C MEX-interfaced codes were developed. These integrator codes are called by MATLAB the same way as the serial MEX codes, but run on NVIDIA GPUs, which greatly speedup computations. We showed that using the GPU codes allowed the codes to run almost one hundred times faster than the serial codes for some cases, and for the typical simulations of vortex rings, around twenty to thirty times faster on an inexpensive GPU card on a single PC. The NLSEmagic code package is distributed on a dedicated website which has already received hundreds of visitors from dozens of countries around the world.

11.2 PUBLICATIONS

The research done in the course of this study have yielded several submitted and in-progress publications. The formulation and testing of the two-step high-order compact scheme (Chapter 3) have been submitted in a paper [137] to the Journal of Computational and Applied Mathematics (JCAM). The development of the modulus-squared Dirichlet boundary condition (Chapter 4) has also been compiled into a paper [138] submitted to JCAM. A paper [139] describing the stability results (Chapter 5) has been submitted to the Journal of Numerical Analysis, Industrial and Applied Mathematics. The development of the NLSEmagic code package (Chapter 8) including all integrators described in Chapters 6 and 7 has been combined into a paper [140] in preparation to be submitted to the Journal of Computer Physics Communications. Further publications are also expected to arise out of the vortex ring dynamical studies (Chapters 9 and 10).

11.3 REPRODUCIBILITY OF RESULTS

As mentioned in Chapter 8, reproduction of research results is very important. Through the provided code package of NLSEmagic and the details written in the figure captions, one can reproduce all the results presented in this dissertation. In order to greatly assist in reproducing the results, we have included on `www.nlsemagic.com` a compressed

file RMC_Dissertation_Figures_Reproduce.zip which contains text files named figure_x_y.txt which have either a direct copy-paste list of parameters and flags to copy into the appropriate NLSExD.m files, or instructions on obtaining the result presented in Fig. x.y. For figures which are not auto-generated in the NLSExD.m files, MATLAB scripts codes are provided which are used to produce the results and compile the data into the figure.

11.4 FURTHER EXTENSIONS

The work presented in this dissertation lends itself to many further extensions. In this final section, we mention some suggestions of possible further work.

As was noted in Chapter 3, in the development of the 2SHOC scheme for two dimensions, the standard five-point central differencing was used in the two-step high-order formulation, rather than the more accurate nine-point stencil. As an extension to the scheme, one could utilize the nine-point stencil and determine if the additional computation is worth the boost in accuracy. Also, analogous enhancements to the scheme in three dimensions could be explored, utilizing the additional grid points available in a three-dimensional compact stencil.

Additional development of the MSD boundary condition is also possible. For example, instead of using only the closest interior grid point to compute the MSD condition, a combination of near-grid points could be used to increase accuracy. Also, it would be interesting to compare the results of using the MSD boundary condition with that of using the radiation boundary conditions mentioned in Chapter 4.

There is an enormous amount of further development possible for the NLSEmagic code package with its integrators. For example, additional numerical schemes could be added to the package (such as mimetic operators [141]) which could then be called from the driver scripts in the same manner as the current integrators. Integrators containing alternative versions of the NLSE (such as two coupled NLSEs, or an NLSE with a saturable or cubic-quintic nonlinearity) could also be produced, using the current

integrators as a model. For use with very large problems, the CUDA integrator codes could be updated to take advantage of multiple GPU cards on a single machine, or even use MPI for use with PC clusters with numerous GPUs. The NLSEmagic driver scripts could also be updated to output the simulation data for post-processing using more advanced visualization procedures.

The study of vortex rings in the NLSE promises to be an area of research for some time to come, especially in the context of BECs. Extensions to the specific studies shown in Chapters 9 and 10 include the interaction and collisions of two vortex rings of unequal radii, equal-charge rings in diagonal offset orientations, and collisions of vortex rings exhibiting quadrupole oscillations, just to name a few. The further study of the stacked multi-ring co-moving solutions shown at the end of Chapter 9, including their underlying structure and dynamics would make a particularly interesting new avenue of research. Additional extensions to the study of vortex rings would be to study dynamics similar to those done in this dissertation, but with the rings situated inside an external harmonic trap, making the results more practical to BEC experimentalists. Another interesting topic to explore would be to find *stable* bright vortex rings in the homogeneous cubic-quintic NLSE in a manner akin to the stabilization of two-dimensional bright vortices in the cubic-quintic NLSE [13, 14]. Finally, another topic of current interest that could be tackled using the efficient methods/codes developed herein is the study of quantum turbulence (see Refs. [142–144]) in three-dimensions whose very essence relies on the interaction, annihilation, and nucleation of vortex ring structures.

It is our hope that through providing the NLSEmagic software package, that many of the research possibilities mentioned here can be readily approached by future students, as well as researchers in the field.

BIBLIOGRAPHY

[1] L. Debnath. *Nonlinear Partial Differential Equations for Scientists and Engineers*. Birkhauser Boston, second edition, 2005.

[2] C. Sulem and P. L. Sulem. *The nonlinear Schrödinger equation, self-focusing and wave collapse*, volume 139 of Appl. Math. Sci. Springer, 1999.

[3] B. A. Malomed. Variational methods in nonlinear fiber-optics and related fields. *Prog. Opt.*, **43** (2002) 71–193.

[4] L. P. Pitaevskii F. Dalfovo, S. Giorgini and S. Stringari. Theory of Bose-Einstein condensation in trapped gases. *Rev. Mod. Phys.*, **71**, 3 (1999) 463–512.

[5] T. Giamarchi, C. Rüegg and O. Tchernyshyov. Bose-Einstein condensation in magnetic insulators. *Nat. Phys.*, **4**, 3 (2008) 198–204.

[6] H. Deng, H. Haug and Y. Yamamoto. Exciton-polariton Bose-Einstein condensation. *Rev. Mod. Phys.*, **82**, 2 (2010) 1489–1537.

[7] J. Klars, J. Schmitt, F. Vewinger and M. Weitz. Bose-Einstein condensation of photons in a 'white-wall' photon box. *J. Phys. Conf.*, **264** (2011) 012005.

[8] H. Saito and M. Ueda. Mean-field analysis of collapsing and exploding Bose-Einstein condensates. *Phys. Rev. A*, **65** (2002) 033624.

[9] S. Wuster, J. J. Hope and C. M. Savage. Collapsing Bose-Einstein condensates beyond the Gross-Pitaevskii approximation. *Phys. Rev. A*, **71** (2005) 033604.

[10] M. Ueda and H. Saito. A consistent picture of a collapsing Bose–Einstein condensate. *J. Phys. Soc. of Jap.*, **72SC**, C (2003) 127–133.

[11] P. A. Altin, G. R. Dennis, G. D. McDonald, D. Döring, J. E. Debs, J. D. Close, C. M. Savage and N. P. Robins. Collapse and three-body loss in a ^{85}Rb Bose-Einstein condensate. *Phys. Rev. A*, **84** (2011) 033632.

[12] P.G. Kevrekidis, D.J. Frantzeskakis and R. Carretero-González. *Emergent Nonlinear Phenomena in Bose-Einstein Condensates: Theory and Experiment*, volume 45. Springer Series on Atomic, Optical, and Plasma Physics, 2008.

[13] R. M. Caplan, Q. E. Hoq, R. Carretero-González and P. G. Kevrekidis. Azimuthal modulational instability of vortices in the nonlinear Schr"odinger equation. *Opt. Comm.*, **282** (2009) 1399–1405.

[14] R. M. Caplan, R. Carretero-González, P.G. Kevrekidis and B.A. Malomed. Existence, stability, and scattering of bright vortices in the cubic-quintic nonlinear Schrödinger equation. *Math. Comp. Sim.*, **82** (2012) 1150–1171.

[15] P. H. Roberts and J. Grant. Motions in a Bose condensate I. The structure of the large circular vortex. *J. Phys. A: Gen. Phys.*, **4** (1971) 55–72.

[16] C. H. Hsueh, S. C. Gou, T. L. Horng and Y. M. Kao. Vortex-ring solutions of the Gross–Pitaevskii equation for an axisymmtrically trapped Bose–Einstein

condensate. *J. Phys. B*, **40**, 24 (2007) 4561–4571.

[17] B. Anderson. Resource article: Experiments with vortices in superfluid atomic gases. *J. Low Temp. Phys.*, **161** (2010) 574–602.

[18] G. W. Rayfield and F. Reif. Quantized vortex rings in superfluid helium. *Phys. Rev.*, **136** (1964) A1194–A1208.

[19] G. Gamota. Creation of quantized vortex rings in superfluid helium. *Phys. Rev. Lett.*, **31** (1973) 517–520.

[20] B. P. Anderson, P. C. Haljan, C. A. Regal, D. L. Feder, L. A. Collins, C. W. Clark and E. A Cornell. Watching dark solitons decay into vortex rings in a Bose-Einstein condensate. *Phys. Rev. Lett.*, **86** (2001) 2926–2929.

[21] S. Levy I. Shomroni, E. Lahoud and J. Steinhauer. Evidence for an oscillating soliton/vortex ring by density engineering of a Bose-Einstein condensate,. *Nat. Phys. Lett.*, **5** (2009) 193–197.

[22] J. Ruostekoski and Z. Dutton. Engineering vortex rings and systems for controlled studies of vortex interactions in Bose-Einstein condensates. *Phys. Rev. A*, **72** (2005) 063626.

[23] J. Ruostekoski and J. R. Anglin. Creating vortex rings and three-dimensional skyrmions in Bose-Einstein condensates. *Phys. Rev. Lett.*, **86**, 18 (2001) 3934–3937.

[24] Naomi S. Ginsberg, Joachim Brand and Lene Vestergaard Hau. Observation of hybrid soliton vortex-ring structures in Bose–Einstein condensates. *Phys. Rev. Lett.*, **94** (2005) 040403.

[25] R. Carretero-González, B. P. Anderson, D. J. Frantzeskakis P. G. Kevrekidis and N. Whitaker. Dynamics of vortex formation in merging Bose-Einstein condensate fragments. *Phys. Rev. A*, **77** (2008) 033625.

[26] R. Carretero-González, N. Whitaker, P. G. Kevrekidis and D. J. Frantzeskakis. Vortex structures formed by the interference of sliced condensates. *Phys. Rev. A*, **77** (2008) 023605.

[27] A. S. Rodrigues, P. G. Kevrekidis, R. Carretero-González, T. J. Alexander D. J. Frantzeskakis, P. Schmelcher and Yu.S. Kivshar. SpinorBose-Einstein condensate past an obstacle. *Phys. Rev. A*, **79** (2009) 043603.

[28] R. G. Scott, A. M. Martin, S. Bujkiewicz, T. M. Fromhold, N. Malossi, O. Morsch, M. Cristiani and E. Arimondo. Transport and disruption of Bose–Einstein condensates in optical lattices. *Phys. Rev. A*, **69** (2004) 033605.

[29] N. G. Berloff and C. F. Barenghi. Vortex nucleation by collapsing bubbles in Bose-Einstein condensates,. *Phys. Rev. Lett.*, **93**, 9 (2004) 090401.

[30] N. G. Berloff. Evolution of rarefaction pulses into vortex rings,. *Phys. Rev. B*, **65** (2002) 174518.

[31] N. G. Berloff. Vortex nucleation by a moving ion in a Bose condensate,. *Phys. Lett. A*, **277**, 9 (2000) 240–244.

[32] N. G. Berloff and P. H. Roberts. Motions in a Bose condensate: VII. boundary-layer separation. *Jour. Phys. A: Math. Gen.*, **33**, 22 (2000) 4025.

[33] N. G Berloff and P. H. Roberts. Motion in a Bose condensate: VIII. The electron bubble. *Jour. Phys. A: Math. Gen.*, **34**, 1 (2001) 81.

[34] N. G. Berloff and P. H Roberts. Motion in a Bose condensate: IX. Crow instability of antiparallel vortex pairs. *J. Phys. A*, **34**, 47 (2001) 10057.

[35] J. F. McCann B. Jackson and C. S. Adams. Vortex rings and mutual drag in trapped Bose-Einstein condensates,. *Phys. Rev. A*, **60**, 6 (1999) 4882–4885.

[36] D. Amit and E. P. Gross. Vortex rings in a Bose fluid,. *Phys. Rev.*, **145**, 1 (1966) 130–136.

[37] A. L. Fetter. Vortices in an imperfect Bose gas. IV. Translational velocity. *Phys. Rev.*, **151** (1966) 100–104.

[38] J. Koplik and H. Levine. Scattering of superfluid vortex rings. *Phys. Rev. Lett.*, **76**, 25 (1996) 4745–4748.

[39] C A Jones and P H Roberts. Motions in a Bose condensate. IV. Axisymmetric solitary waves. *Jour. Phys. A: Math. Gen.*, **15**, 8 (1982) 2599.

[40] N. G. Berloff. Interactions of vortices with rarefaction solitary waves in a Bose–Einstein condensate and their role in the decay of superfluid turbulence. *Phys. Rev. A*, **69** (2004) 053601.

[41] M. Abad, M. Guilleumas, R. Mayol and M. Pi. Vortex rings in toroidal Bose-Einstein condensates. *Las. Phys.*, **18**, 5 (2008) 648–652.

[42] M. Guilleumas, D. M. Jezeka, M. Pi R. Mayol and M. Barranco1. Generating vortex rings in Bose-Einstein condensates in the line-source approximation. *Phys. Rev. A*, **65** (2002) 053609.

[43] L. C. Crasovan, V. M. Pérez-Gárcía, I. Danaila, D. Mihalache and L. Torner. Three-dimensional parallel vortex rings in Bose-Einstein condensates,. *Phys. Rev. A*, **70** (2004) 033605.

[44] S. C. Gou T. L. Horng and T. C. Lin. Bending-wave instability of a vortex ring in a trapped Bose-Einstein condensate. *Phys. Rev. A*, **74** (2006) 041603(R).

[45] C.-H. Hsueh T.-L. Horng and S.-C. Gou. Transition to quantum turbulence in a Bose-Einstein condensate through the bending-wave instability of a single-vortex ring,. *Phys. Rev. A*, **77** (2008) 063625.

[46] J. F. McCann B. Jackson and C. S. Adams. Vortex line and ring dynamics in trapped Bose-Einstein condensates. *Phys. Rev. A*, **61** (1999) 013604.

[47] S. Komineas and J. Brand. Collisions of solitons and vortex rings in cylindrical Bose-Einstein condensates. *Phys. Rev. Lett*, **95** (2005) 110401.

[48] S. Komineas. Vortex rings and solitary waves in trapped Bose–Einstein condensates. *Eur. Phys. Jour.- Spec. Top.*, **147** (2007) 133–152.

[49] H. Levine, personal communication.

[50] R. J. Donnelly. *Quantized Vortices in Helium II*. Number 3 in Cambridge Studies in Low Temperature Physics. Cambridge University Press, 1991.

[51] V. M. Prez-García and X. Liu. Numerical methods for the simulation of trapped nonlinear Schrödinger systems. *Appl. Math. and Comp.*, **144**, 2-3 (2003) 215 – 235.

[52] W. F. Spotz and G. F. Carey. Iterative and parallel performance of high-order compact systems. *SIAM J. Sci. Comp.*, **19**, 1 (1998) 1–14.

[53] J. Zhang. An explicit fourth-order compact finite difference scheme for three-dimensional convection-diffusion equation. *Comm. Num. Meth. Engi.*, **14** (1998) 209–218.

[54] W. F. Spotz and G. F. Carey. A high-order compact formulation for the 3D Poisson equation. *Num. Meth. Parti. Diff. Equ.*, **12** (1996) 235–243.

[55] W. F. Spotz and G. F. Carey. High-order compact scheme for the steady stream-function vorticity equations. *Inter. Jour. Num. Meth. Engi.*, **38**, 20 (1995) 3497–3512.

[56] L. Ge and J. Zhang. Symbolic computation of high-order compact difference schemes for three-dimensional linear elliptic partial differential equations with variable coefficients. *J. Comp. Appl. Math.*, **143** (2002) 9–27.

[57] W. Liao. An implicit fourth-order compact finite difference scheme for one-dimensional Burgers' equation. *App. Math. Comp.*, **206** (2008) 755–764.

[58] M. Ciment and S. H. Leventhal. Higher order compact implicit schemes for the wave equation. *Math. Comp.*, **29**, 132 (1975) 985–994.

[59] S. Abarbanel and A. Kumar. Compact high-order schemes for the Euler equations. *Jour. Sci. Comp.*, **3** (1988) 275–288.

[60] W. F. Spotz and G. F. Carey. Extension of high-order compact schemes to time-dependent problems. *Num. Meth. Parti. Diff. Equ.*, **17**, 6 (2001) 657–672.

[61] Jiten C. Kalita, D. C. Dalal and Anoop K. Dass. A class of higher order compact schemes for the unsteady two-dimensional convection-diffusion equation with variable convection coefficients. *Inter. Jour. Num. Meth. Fluids*, **38**, 12 (2002) 1111–1131.

[62] A. Mohebbi and M. Dehghan. The use of compact boundary value method for the solution of two-dimensional Schrödinger equation. *J. Comp. Appl. Math.*, **225** (2009) 124–134.

[63] J. C. Kalita, P. Chhabra and S. Kumar. A semi-discrete higher order compact scheme for the unsteady two-dimensional Shrödinger equation. *J. Comp. Appl. Math.*, **197** (2006) 141–149.

[64] S. Xie, G. Li and S. Yi. Compact finite-difference schemes with high accuracy for one-dimensional nonlinear Schrödinger equation. *Comp. Meth. Appl. Mech. Engi.*, **198** (2009) 1052–1060.

[65] M. Dehghan and A. Taleei. A compact split-step finite difference method for solving the nonlinear Schrödinger equations with constant and variable coefficients.

Comp. Phys. Comm., **181** (2010) 43–51.

[66] J. C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. SIAM, second edition, 2004.

[67] B. Fornberg. *A Practical Guide to Pseudospectral Methods*. Cambridge University Press, 1996.

[68] M. H. Carpenter, D. Gottlieb and S. Abarbanel. Time-stable boundary conditions for finite-difference schemes solving hyperbolic systems: Methodology and application to high-order compact schemes. *Jour. of Comp. Phys.*, **111**, 2 (1994) 220 – 236.

[69] M. H. Carpenter, D. Gottlieb and S. Abarbanel. Stable and accurate boundary treatments for compact, high-order finite-difference schemes. *Appl. Num. Math.*, **12**, 1-3 (1993) 55 – 87.

[70] W. H. Raymond and H. L. Kuo. A radiation boundary condition for multi-dimensional flows. *Q. Journ. of the Roy. Met. Soc.*, **110**, 464 (1984) 535–551.

[71] X. Antoine, A. Arnold, C. Besse, M. Ehrhardt and A. Schädle. A review of transparent and artificial boundary conditions techniques for linear and nonlinear Schrödinger equations. *Comm. in Comp. Phys.*, **4**, 4 (2008) 729–796.

[72] Dale R. Durran. Open boundary conditions: Fact and fiction. In P. F. Hodnett, editor, *IUTAM Symposium on Advances in Mathematical Modelling of Atmosphere and Ocean Dynamics*, volume 61 of *Fluid Mechanics and Its Applications*, pages 1–18. Springer, 2001.

[73] S.E. Krakiwsky, L.E. Turner and M.M. Okoniewski. Graphics processor unit GPU acceleration of finite-difference time-domain (FDTD) algorithm. In *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, volume 5, pages 265–268, 2004.

[74] S. Adams, J. Payne and R. Boppana. Finite difference time domain (FDTD) simulations using graphics processors. *HPCMP Usr. Grp. Conf.*, (2007) 334–338.

[75] A. Balevic, L. Rockstroh, A. Tausendfreund, S. Patzelt, G. Goch and S. Simon. Accelerating simulations of light scattering based on finite-difference time-domain method with general purpose GPUs. In *Computational Science and Engineering, 2008. CSE '08*, pages 327–334, 2008.

[76] P. Micikevicius. 3D finite difference computation on GPUs using CUDA. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, pages 79–84, 2009.

[77] David Michéa and Dimitri Komatitsch. Accelerating a three-dimensional finite-difference wave propagation code using GPU graphics cards. *Geo. J. Inter.*, **182**, 1 (2010) 389–402.

[78] K. A. Hawick and D. P. Playne. Numerical simulation of the complex Ginzburg-Landau equation on GPUs with CUDA. *Massey University*, (2010) CSTN–070.

[79] P. G. Drazin and R. S. Johnson. *Solitons: An Introduction*. Number 2 in Cambridge Texts in Applied Mathematics. Cambridge University Press, 1989.

[80] Y. S. Kivshar and B. Luther-Davies. Dark optical solitons: physics and applications. *Phys. Rep.*, **298**, 2-3 (1998) 81–197.

[81] E.A. Spiegel. Fluid dynamical form of the linear and nonlinear Schrödinger equations. *Phys. D: Nonl. Phen.*, **1**, 2 (1980) 236–240.

[82] K. Kasamatsu and M. Tsubota. Quantized vortices in atomic Bose-Einstein condensates. In *Quantum Turbulence*, volume 16 of *Progress in Low Temperature Physics*, pages 351–403. Elsevier, 2009.

[83] D.J. Frantzeskakis R. Carretero-González and P.G. Kevrekidis. Nonlinear waves in Bose-Einstein condensates: physical relevance and mathematical techniques. *Nonlinearity*, **21** (2008) R139–R202.

[84] R. Carretero-González P. T. Torres, P. Schmelcher S. Middelkamp and D. J. Frantzeskakis P. G. Kevrekidis. Vortex interaction dynamics in trapped Bose-Einstein condensates. *Comm. Pure Appl. Ana.*, **10** (2011) 1589–1615.

[85] G. Herring, L. D. Carr, R. Carretero-González, P. G. Kevrekidis and D. J. Frantzeskakis. Radially symmetric nonlinear states of harmonically trapped Bose–Einstein condensates. *Phys. Rev. A*, **77** (2008) 023625.

[86] L. P. Pitaevskii. Vortex lines in an imperfect Bose gas. *Sov. Phys. JETP*, **13**, 2 (1961) 451–454.

[87] A. L. Fetter. Vortices in an imperfect Bose gas. I. The condensate. *Phys. Rev.*, **138** (1965) A429–A437.

[88] M. P. Kawatra and R. K. Pathria. Quantized vortices in an imperfect Bose gas and the breakdown of superfluidity in liquid Helium II. *Phys. Rev.*, **151** (1966) 132–137.

[89] D. Mihalache, D. Mazilu, L.-C. Crasovan, I. Towers, A. V. Buryak, B. A. Malomed, L. Torner, J. P. Torres and F. Lederer. Stable spinning optical solitons in three dimensions. *Phys. Rev. Lett.*, **88** (2002) 073902.

[90] V. L. Ginzburg and L. P. Pitaevskii. On the theory of superfluidity. *Sov. Phys. JETP*, **7** (1958) 858–861.

[91] P. Roberts and N. Berloff. The nonlinear Schrödinger equation as a model of superfluidity. In C. Barenghi, R. Donnelly and W. Vinen, editors, *Quantized Vortex Dynamics and Superfluid Turbulence*, volume 571 of *Lecture Notes in Physics*, pages 235–257. Springer Berlin / Heidelberg, 2001.

[92] P. G. Saffman. *Vortex Dynamics*. Cambridge Monographs on Mechanics. Cambridge University Press, 1995.

[93] M. S. Ismail. A fourth-order explicit schemes for the coupled nonlinear Schrödinger equation. *Appli. Math. Comp.*, **196** (2008) 273–284.

[94] M. M. Cerimele, M. L. Chiofalo, F. Pistella, S. Succi and M. P. Tosi. Numerical solution of the Gross-Pitaevskii equation using an explicit finite-difference scheme: An application to trapped Bose-Einstein condensates. *Phys. Rev. E.*, **62**, 1 (2000)

1382–1389.

[95] J.C. Butcher. A history of Runge-Kutta methods. *Appl. Num. Math.*, **20**, 3 (1996) 247–260.

[96] D. J. Fyfe. Economical evaluation of Runge-Kutta formula. *Math. Comp.*, **20** (1966) 392–398.

[97] M. H. Carpenter and C. A. Kennedy. Fourth-order $2N$-storage Runge-Kutta schemes. *NASA Tech. Mem.*, (1994).

[98] P. Muruganandam and S.K. Adhikari. Fortran programs for the time-dependent Gross-Pitaevskii equation in a fully anisotropic trap. *Comp. Phys. Comm.*, **180**, 10 (2009) 1888 – 1912.

[99] W. Milne. *Numerical Solution of Differential Equations*. Dover Publications Inc., second edition, 1970.

[100] W. F. Spotz. *High-order compact finite difference schemes for computational mechanics*. PhD thesis, University of Texas at Austin, 1995.

[101] R. Carretero, personal communication.

[102] P.G. Kevrekidis, D.J. Frantzeskakis R. Carretero-González and I.G. Kevrekidis. Vortices in Bose-Einstein condensates: Some recent developments. *Mod. Phys. Lett. B*, **18**, 30 (2004) 1481–1505.

[103] J. D. Lambert. *Numerical Methods for Ordinary Differential Systems*. John Wiley & Sons, 1991.

[104] W. Dai. An unconditionally stable three-level explicit difference scheme for the Schrödinger equation with a variable coefficient. *SIAM Jour. Num. Analy.*, **29**, 1 (1992) 174–181.

[105] R. Bronson. *Schaum's Outline of Matrix Operations*. McGraw-Hill, first edition, 1988.

[106] G. H. Golub. *Matrix Computations*. Johns Hopkins University Press, third edition, 1996.

[107] I. Kra and S. R. Simanca. On circulant matrices. *Not. AMS*, **59**, 3 (2012) 368–377.

[108] T.A. Davis. *MATLAB Primer, Eighth Edition*. CRC Press, 2010.

[109] NVIDIA. CUDA C programming guide 4.0. *http://developer.nvidia.com/*, (2011).

[110] NVIDIA. White paper: NVIDIAs next generation CUDA compute architecture: Fermi. *http://www.nvidia.com*, (2009).

[111] NVIDIA. Why choose tesla. *http://www.nvidia.com/*, (2011).

[112] NVIDIA. Fermi compatibility guide for CUDA applications 1.5. *http://developer.nvidia.com/*, (2011).

[113] John E. Stone, David Gohara and Guochun Shi. OpenCL: A parallel programming standard for heterogeneous computing systems. *Comp. in Sci. and Engi.*, **12** (2010) 66–73.

[114] Khronos Group. OpenCL overview. *http://www.khronos.org/opencl/*, (2011).

[115] K. Komatsu, K. Sato, Y. Arai, K. Koyama, H. Takizawa and H. Kobayashi. Evaluating performance and portability of OpenCL programs. *The Fifth International Workshop on Automatic Performance Tuning (iWAPT2010)*, (2010).

[116] J. Fang, A. L. Varbanescu and H. Sips. A comprehensive performance comparison of CUDA and OpenCL. In *Parallel Processing (ICPP), 2011*, pages 216–225, 2011.

[117] K. Karimi, N. G. Dickson and F. Hamze. A performance comparison of CUDA and OpenCL. *ArXiv e-prints*, (2010) arXiv:1005.2581.

[118] G. Martinez, W. Feng and M. Gardner. CU2CL: A CUDA-to-OpenCL translator for multi- and many-core architectures. Technical report, Virginia Tech, 2011.

[119] The Portland Group. PGI CUDA-x86: CUDA programming for multi-core CPUs. *PGInsider*, **2**, 4 (2010) a1.

[120] J. Stratton, S. Stone and W. Hwu. *MCUDA: An Efficient Implementation of CUDA Kernels for Multi-core CPUs*, volume 5335 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2008.

[121] J. S. Vetter, R. Glassbrook, J. Dongarra, K. Schwan, B. Loftis, S. McNally, J. Meredith, J. Rogers, P. Roth, K. Spafford and S. Yalamanchili. Keeneland: Bringing heterogeneous GPU computing to the computational science community. *Comp. in Sci. and Engi.*, **13**, 5 (2011) 90–95.

[122] The Portland Group. PGI accelerator compilers. *http://www.pgroup.com/resources/accel.htm*, (2011).

[123] Mathworks. MATLAB GPU computing with NVIDIA CUDA-enabled GPUs. *http://www.mathworks.com/discovery/matlab-gpu.html*, (2011).

[124] NVIDIA. White paper: Accelerating MATLAB with CUDA using MEX files. *http://developer.download.nvidia.com*, (2007).

[125] B. Jang, D. Schaa, P. Mistry and D. Kaeli. Exploiting memory access patterns to improve memory performance in data-parallel architectures. *IEEE Trans. on Par. and Dist. Sys.*, **22**, 1 (2011) 105–118.

[126] D. E. Post and L. G. Votta. Computational science demands a new paradigm. *Phys. Today*, **58**, 1 (2005) 35–41.

[127] D. L. Donoho, A. Maleki, I. U. Rahman, M. Shahram and V. Stodden. Reproducible research in computational harmonic analysis. *Comp. Sci. Engi.*, **11**, 1 (2009) 8–18.

[128] K. Diethelm. The limits of reproducibility in numerical simulation. *Comp. in Sci. Engi.*, **14**, 1 (2012) 64–72.

[129] O. Woodford and J. Conti. vol3d: 3D volume (voxel) rendering. *MATLAB Central File Exchange*, (2011).

[130] C. T. Kelley. *Solving Nonlinear Equations with Newton's Method*. Fundamentals of Algorithms. SIAM, 2003.

[131] F. Moisy. Ezyfit: A free curve-fitting toolbox for MATLAB.

http://www.fast.u-psud.fr/ezyfit/, (2010).

[132] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, second edition, 2006.

[133] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comp.*, **7**, 3 (1986) 856–869.

[134] K. Mohseni and T. Colonius. Numerical treatment of polar coordinate singularities. *J. Comput. Phys.*, **157**, 2 (2000) 787–795.

[135] J. L. Helm, C. F. Barenghi and A. J. Youd. Slowing down of vortex rings in Bose–Einstein condensates. *Phys. Rev. A*, **83** (2011) 045601.

[136] B. N. Shashikanth and J. E. Marsden. Leapfrogging vortex rings: Hamiltonian structure, geometric phases and discrete reduction. *Fld. Dyn. Res.*, **33**, 4 (2003) 333–356.

[137] R. M. Caplan and R. Carretero-González. A two-step high order compact scheme for the Laplacian operator and its implementation in a fully explicit method for integrating the nonlinear Schrödinger equation. *Submitted to Appl. Math and Comp.*, (2012) arXiv:1109.1027.

[138] R. M. Caplan and R. Carretero-González. A modulus-squared Dirichlet boundary condition for time-dependant complex partial differential equations and its application to the nonlinear Schrödinger equation. *Submitted to Appl. Math and Comp.*, (2012) arXiv:1110.0569.

[139] R. M. Caplan and R. Carretero-González. Numerical stability of explicit Runge-Kutta finite-difference schemes for the nonlinear Schrödinger equation. *Submitted to J. Num. Analy. Indus. App. Math.*, (2012) arXiv:1107.4810.

[140] R. M. Caplan. NLSEmagic: Nonlinear Schrödinger equation multidimensional Matlab-based GPU-accelerated integrators using compact high-order schemes. *In preperation to be submitted to Comp. Phys. Comm.*, (2012) arXiv:1203.1263.

[141] J.E. Castillo, J.M. Hymanb, M. Shashkov and S. Steinberg. Fourth- and sixth-order conservative finite difference approximations of the divergence and gradient. *Appl. Num. Math.*, **37** (2001) 171187.

[142] C. Raman, J. R. Abo-Shaeer, J. M. Vogels, K. Xu and W. Ketterle. Vortex nucleation in a stirred Bose-Einstein condensate. *Phys. Rev. Lett.*, **87** (2001) 210402.

[143] E. A. L. Henn, J. A. Seman, G. Roati, K. M. F. Magalhães and V. S. Bagnato. Emergence of turbulence in an oscillating Bose-Einstein condensate. *Phys. Rev. Lett.*, **103** (2009) 045301.

[144] C. F. Barenghi, R. J. Donnelly and W. F. Vinen. *Quantized Vortex Dynamics and Superfluid Turbulence*, volume 571 of *Lecture Notes in Physics*. Springer, 2001.

APPENDIX A

Derivation and Computation of the Vortex Core Parameter

Derivation and Computation of the Vortex Core Parameter

The 'vortex core parameter' L_0 is defined as the convergent part of the energy per unit length of a dark vortex line in the NLSE chosen here to be in the form

$$i\Psi_t + a\nabla^2\Psi + s|\Psi|^2\Psi = 0,$$

where $\Psi = \Psi \exp[i\Omega\, t]$ and Ω is the frequency. First, we write the energy per unit length (as a departure from the uniform background density $\Psi_\infty = \Omega/s$) of a solution Ψ of the NLSE which does not change in the z-direction in cylindrical coordinates as [91]

$$\mathcal{E} = \int_0^{2\pi}\int_0^{\infty}\left[a\,|\nabla\Psi|^2 - \frac{s}{2}\left(\frac{\Omega}{s} - |\Psi|^2\right)^2\right] r\,dr\,d\theta. \tag{A.1}$$

where

$$|\nabla\Psi|^2 = |\Psi_r|^2 + \frac{1}{r^2}|\Psi_\theta|^2.$$

Inserting the vortex solution

$$\Psi = f(r)\exp[i(m\,\theta + \Omega\, t)]$$

into Eq. (A.1) yields

$$\frac{\mathcal{E}}{2\pi} = a\int_0^{\infty}\left(\frac{df}{dr}\right)^2 r\,dr - \frac{s}{2}\int_0^{\infty}\left(\frac{\Omega}{s} - f^2(r)\right)^2 r\,dr + a\,m^2\int_0^{\infty}\frac{f^2(r)}{r}\,dr, \tag{A.2}$$

where $f(r)$ is found numerically by solving the ODE (see Sec. 9.1.1)

$$-\left(\Omega + \frac{a\,m^2}{r^2}\right)f(r) + a\left(\frac{1}{r}\frac{df}{dr} + \frac{d^2f}{dr^2}\right) + s\,f^3(r) = 0. \tag{A.3}$$

The first two integrals of Eq. (A.2) converge but the third diverges. Focusing on the third integral, by integration-by-parts we have

$$\int_0^\infty \frac{f^2(r)}{r}\,dr = f^2(r)\ln(r)\big|_0^\infty - 2\int_0^\infty f(r)\,\frac{df}{dr}\ln(r)\,dr, \tag{A.4}$$

where the integral in Eq. (A.4) converges. Since by definition, the vortex has zero-density at the core, $f(0) = 0$ and by defining a cut-off distance L, Eq. (A.4) becomes

$$\int_0^\infty \frac{f^2(r)}{r}\,dr \approx f^2(L)\ln(L) - 2\int_0^\infty f(r)\,\frac{df}{dr}\ln(r)\,dr. \tag{A.5}$$

By using the asymptotic profile given by Eq. (2.25),

$$\frac{\Omega}{s} - f^2(L) \approx \frac{\Omega}{s}\left(\frac{r_c}{L}\right)^2,$$

where $r_c = \sqrt{-a\,m^2/\Omega}$ is the approximate width of the vortex core. Therefore for large L/r_c, $f^2(L) \approx \Omega/s$. The energy can now be written as

$$\begin{aligned}\frac{\mathcal{E}}{2\pi} =& a\int_0^\infty \left(\frac{df}{dr}\right)^2 r\,dr - \frac{s}{2}\int_0^\infty \left(\frac{\Omega}{s} - f^2(r)\right)^2 r\,dr \\ &- 2\,a\,m^2\int_0^\infty f(r)\,\frac{df}{dr}\ln(r)\,dr + a\,m^2\frac{\Omega}{s}\ln(L).\end{aligned} \tag{A.6}$$

Combining the convergent integrals and adding and subtracting a $\ln(r_c)$ term yields

$$\mathcal{E} = \frac{2\,\pi\,a\,m^2\,\Omega}{s}\left[\ln\left(\frac{L}{r_c}\right) + L_0(m)\right], \tag{A.7}$$

where L_0 is called the 'vortex core parameter' and is found by numerically computing

$$\begin{aligned}L_0(m) =& \frac{s}{a\,m^2\,\Omega}\left(a\int_0^\infty \left(\frac{df}{dr}\right)^2 r\,dr - \frac{s}{2}\int_0^\infty \left(\frac{\Omega}{s} - f^2(r)\right)^2 r\,dr\right. \\ &\left. -\,2\,a\,m^2\int_0^\infty f(r)\,\frac{df}{dr}\ln(r)\,dr\right) + \ln(r_c),\end{aligned} \tag{A.8}$$

where we note that due to the addition of the terms involving r_c, for a given m, $L_0(m)$ is invariant over the parameters a, s, and Ω.

Another way to compute L_0 which is similar to Eq. (A.8) can be found by simply initially assuming that $f^2(r \geq r_c) \approx \Omega/s$, in which case the divergent integral is split into two pieces yielding the same energy expression, but with the computation of L_0 as

$$L_0(m) = \frac{s}{a\, m^2\, \Omega} \left(a \int_0^\infty \left(\frac{df}{dr} \right)^2 r\, dr - \frac{s}{2} \int_0^\infty \left(\frac{\Omega}{s} - f^2(r) \right)^2 r\, dr + a\, m^2 \int_0^\infty \frac{f^2(r)}{r}\, dr \right) - \ln\left(\frac{L}{r_c} \right), \tag{A.9}$$

We compute the integrals in Eqs. (A.8) and (A.9) using a simple trapezoid integration on a numerically-exact $f(r)$ computed on a fine mesh using the methods described in Chapter 9. Through numerical testing, we have found that using Eq. (A.9) gives better results using a smalled domain than Eq. (A.8), but that given a large enough domain, both yield the same results. The cut-off distance L is set to be the maximum r value over the computational domain, which is set to be $L = 500$. The spatial step size for $f(r)$ is set to $h = 0.0075$ and the NLSE parameters used are $a = 1$, $s = -1$, and $\Omega = -1$. For $|m| = 1$, we recover the value $L_0(1) \approx 0.3809$. The values of L_0 for charges $m = 1, ..., 10$ are given in the main text in Table. 2.1 of Chapter 2. We note that the values of L_0 for $|m| > 3$ have not been previously reported, and those for $m = 2$ and $m = 3$ are given with much more accuracy than reported in Ref. [90].

By fitting a linear function to the log–log plot of $L_0(m)$, a relationship between the L_0 values of different charges can be approximated as

$$L_0(m) \approx \frac{L_0(1)}{m^{1.6}},$$

which may indicate that with a different choice of r_c, an $L_0(m)$ could be found that would be equivalent for any m, but such an investigation is beyond the current scope.

In the analysis above, we used the approximation $f^2(L) \approx \Omega/s$. However, for a more accurate energy, one could use the asymptotic profile of Eq. (2.25) instead. Doing so would add an extra small term of order $O\left((r_c/L)^2\right)$. However, since the numerical values of the integrals are very sensitive, it is expected that the error in the integrals outweighs the advantages of the added term.

APPENDIX B

Derivation of a General form of Co-moving Solutions to the NLSE

Derivation of a General form of Co-moving Solutions to the NLSE

Here we show the derivation of a general form for co-moving steady state solutions to the NLSE. We show the formulation for three dimensions, which can be trivially altered for use with lower dimensions.

We start with the NLSE written as

$$i\Psi_t + a\nabla^2\Psi + s|\Psi|^2\Psi = 0, \qquad \Psi = \Psi(x, y, z, t). \tag{B.1}$$

Defining the co-moving variables

$$X = x - v_x t \qquad Y = y - v_y t \qquad Z = z - v_z t, \tag{B.2}$$

in the co-moving frame, the NLSE is

$$i\Phi_t + A\nabla^2\Phi + S|\Phi|^2\Phi = 0, \qquad \Phi = \Phi(X, Y, Z, t), \tag{B.3}$$

where A and S are constants which may be different than a and s. The steady-state solution to Eq. (B.3) is given by

$$\Phi = e^{i\Omega t}U(X, Y, Z), \tag{B.4}$$

in which case Eq. (B.3) yields the time-independent PDE

$$-\Omega\, U + A\nabla^2 U + S|U|^2 U = 0. \tag{B.5}$$

Given a solution $U(X,Y,Z)$ which solves Eq. (B.5), since at time $t=0$, $|\Psi|^2 = |U|^2$, the solution is assumed to have the form

$$\Psi(x,y,t) = e^{i\rho(x,y,z,t)}\, U(X,Y,Z). \tag{B.6}$$

Inserting Eq. (B.6) into Eq. (B.1) yields

$$\begin{aligned} i\,(U_t + i\rho_t U) + a[(-\rho_x^2 + i\rho_{xx})U + 2\,U_x(i\rho_x) + U_{xx} + \\ (-\rho_y^2 + i\rho_{yy})U + 2\,U_y(i\rho_y) + U_{yy} + \\ (-\rho_z^2 + i\rho_{zz})U + 2\,U_z(i\rho_z) + U_{zz}] + s|U|^2\,U = 0. \end{aligned} \tag{B.7}$$

Noting that

$$\begin{aligned} X_x = 1 \qquad X_{xx} = 0 \qquad X_t = -v_x \\ Y_y = 1 \qquad Y_{yy} = 0 \qquad Y_t = -v_y \\ Z_z = 1 \qquad Z_{zz} = 0 \qquad Z_t = -v_z, \end{aligned} \tag{B.8}$$

it follows that

$$\begin{aligned} U_x = U_X \qquad\qquad U_{xx} = U_{XX} \\ U_y = U_Y \qquad\qquad U_{yy} = U_{YY} \\ U_z = U_Z \qquad\qquad U_{zz} = U_{ZZ} \\ U_t = -v_x\,U_X - v_y\,U_Y - v_z\,U_Z. \end{aligned} \tag{B.9}$$

By using the substitutions of Eq. (B.9) and collecting terms, Eq. (B.7) can be put into a form resembling the steady-state form of the NLSE for $U(X,Y,Z)$ given in Eq. (B.5):

$$\begin{aligned}\left[-\rho_t + a\left(-(\rho_x^2+\rho_y^2+\rho_z^2)+i(\rho_{xx}+\rho_{yy}+\rho_{zz})\right)\right]U + a\nabla^2 U + s|U|^2 U = \\ -i\left[(-v_x+2a\rho_x)U_X + (-v_y+2a\rho_y)U_Y + (-v_z+2a\rho_z)U_Z\right],\end{aligned} \tag{B.10}$$

where we have set $S = s$ and $A = a$. In order for Eq. (B.10) to be true given the definition of U (and that the frequency Ω is real-valued) we must require that

$$\begin{aligned} -v_x + 2a\rho_x &= 0, \\ -v_y + 2a\rho_y &= 0, \\ -v_z + 2a\rho_z &= 0, \\ \text{Im}\left[-\rho_t + a\left(-(\rho_x^2+\rho_y^2+\rho_z^2)+i(\rho_{xx}+\rho_{yy}+\rho_{zz})\right)\right] &= 0, \\ \text{Re}\left[-\rho_t + a\left(-(\rho_x^2+\rho_y^2+\rho_z^2)+i(\rho_{xx}+\rho_{yy}+\rho_{zz})\right)\right] &= -\Omega. \end{aligned} \tag{B.11}$$

Integrating the first three equations of Eq. (B.11) yields

$$\begin{aligned} \rho &= \frac{v_x}{2a}\,x + f_1(y,z,t), \\ \rho &= \frac{v_y}{2a}\,y + f_2(x,z,t), \\ \rho &= \frac{v_z}{2a}\,z + f_3(x,y,t), \end{aligned} \tag{B.12}$$

where f_1, f_2, and f_3 are arbitrary functions. For all three equations in Eq. (B.12) to be valid, it is required that

$$\rho(x,y,z,t) = \frac{v_x}{2a}\,x + \frac{v_y}{2a}\,y + \frac{v_z}{2a}\,z + C(t), \tag{B.13}$$

where $C(t)$ is a function of time to be determined. The fourth equation in Eq. (B.11) can now be rewritten as

$$\text{Im}\left[-C'(t) - \frac{v_x^2 + v_y^2 + v_z^2}{4a}\right] = 0,$$

in which case (since all velocities are real), $C'(t) \in \Re$. Therefore, the last equation in Eq. (B.11) gives

$$C'(t) + \frac{v_x^2 + v_y^2 + v_z^2}{4a} = \Omega,$$

which can be integrated to yield

$$C(t) = \left[\Omega - \frac{v_x^2 + v_y^2 + v_z^2}{4a}\right] t + \phi, \tag{B.14}$$

where ϕ is an arbitrary constant (which in this case ends up being the phase shift). Inserting Eq. (B.14) into Eq. (B.13) yields

$$\rho(x, y, z, t) = \frac{v_x}{2a}\, x + \frac{v_y}{2a}\, y + \frac{v_z}{2a}\, z + \left[\Omega - \frac{v_x^2 + v_y^2 + v_z^2}{4a}\right] t + \phi. \tag{B.15}$$

The final result can be summarized as follows: A co-moving steady-state solution to the NLSE defined as

$$i\Psi_t + a\nabla^2\Psi + s|\Psi|^2\Psi = 0, \qquad \Psi = \Psi(x, y, z, t)$$

is given by

$$\Psi(x, y, z, t) = e^{i\rho(x,y,z,t)}\, U(X, Y, Z),$$

where

$$\rho(x, y, z, t) = \frac{v_x}{2a}\, x + \frac{v_y}{2a}\, y + \frac{v_z}{2a}\, z + \left[\Omega - \frac{v_x^2 + v_y^2 + v_z^2}{4a}\right] t + \phi,$$

where ϕ is an arbitrary phase shift and

$$X = x - v_x\, t \qquad Y = y - v_y\, t, \qquad Z = z - v_z\, t,$$

and $U(X, Y, Z)$ solves the time-independent PDE

$$-\Omega\, U + a\nabla^2 U + s|U|^2 U = 0.$$

The formulation presented here can be used to turn a steady-state solution in Ψ into a co-moving solution by multiplying the appropriate exponential terms. Also, if one has a solution to the NLSE which is intrinsically a co-moving steady-state solution, then one could multiply the solution by an appropriate exponential term to make the solution steady-state in $|\Psi|^2$.

APPENDIX C

Proof that Double-Differencing Yields a Fourth-Order Accurate Scheme for the Second Derivative

Proof that Double-Differencing Yields a Fourth-Order Accurate Scheme for the Second Derivative

In order to show that the double-differencing described in Chapter 3 does indeed lead to a fourth-order accurate representation of Ψ_{xx}, it is necessary to prove that $\delta_x^2(\delta_x^2\Psi)$ is an $O(h^2)$ approximation to Ψ_{xxxx}.

We first note that

$$\Psi_{xxxx} = (\Psi_{xx})_{xx} = \left[\delta_x^2\Psi - \frac{h^2}{12}\Psi_{xxxx} + O(h^4)\right]_{xx}.$$

Distributing the second derivative over the differencing and error terms yields

$$\Psi_{xxxx} = (\delta_x^2\Psi)_{xx} - \frac{h^2}{12}\Psi_{xxxxxx} + O(h^4)_{xx}.$$

We now approximate the remaining two spatial derivatives of Ψ with a second-order central-differencing to yield

$$\Psi_{xxxx} = \delta_x^2(\delta_x^2\Psi) - \frac{h^2}{12}(\delta_x^2\Psi)_{xxxx} + O(h^2).$$

Explicitly writing out the differencing of the truncation term gives

$$\Psi_{xxxx} = \delta_x^2(\delta_x^2\Psi) - \frac{1}{12}(\Psi^{i+1}_{xxxx} - 2\Psi^{i}_{xxxx} + \Psi^{i-1}_{xxxx}) + O(h^2),$$

in which case, noting that

$$\Psi_{xxxxxx} = \frac{\Psi^{i+1}_{xxxx} - 2\Psi^{i}_{xxxx} + \Psi^{i-1}_{xxxx}}{h^2} + O(h^2),$$

the truncation term can be rewritten as

$$\Psi_{xxxx} = \frac{\Psi_{i+2} - 4\Psi_{i+1} - 6\Psi_n - 4\Psi_{i-1} + \Psi_{i-2}}{h^4} - \left[\frac{h^2}{12}\Psi_{xxxxxx} + O(h^4)\right] + O(h^2).$$

After collecting terms, this yields

$$\Psi_{xxxx} = \frac{\Psi_{i+2} - 4\Psi_{i+1} - 6\Psi_n - 4\Psi_{i-1} + \Psi_{i-2}}{h^4} + O(h^2),$$

which is the standard 5-point second-order finite-difference for Ψ_{xxxx}. Therefore, the double-differencing *is* equivalent to a second-order approximation of Ψ_{xxxx}, in which case it can be used to formulate a fourth-order accurate approximation to Ψxx as shown in Chapter 3.

APPENDIX D

Proof that No Fourth-Order Nine-Point Compact Two-Dimensional Laplacian Operator Exists

Proof that No Fourth-Order Nine-Point Compact Two-Dimensional Laplacian Operator Exists

We would like to find a compact, nine-point Laplacian operator which is fourth-order accurate, namely

$$\nabla^2\Psi_{i,j} = \frac{1}{h^2}\begin{array}{|c|c|c|}\hline \gamma & \beta & \gamma \\ \hline \beta & \alpha & \beta \\ \hline \gamma & \beta & \gamma \\ \hline \end{array}\,\Psi_{i,j} + O(h^4), \tag{D.1}$$

where $\alpha \neq 0$, $\beta \neq 0$, and $\gamma \neq 0$. Taylor expanding each grid point in Eq. (D.1) yields

$$\begin{aligned}
\Psi_{i+1,j} =& \Psi + h\Psi_x + \frac{h^2}{2}\Psi_{xx} + \frac{h^3}{6}\Psi_{xxx} + \frac{h^4}{24}\Psi_{xxxx} + \frac{h^5}{120}\Psi_{xxxxx} + O(h^6) \qquad \text{(D.2)}\\
\Psi_{i-1,j} =& \Psi - h\Psi_x + \frac{h^2}{2}\Psi_{xx} - \frac{h^3}{6}\Psi_{xxx} + \frac{h^4}{24}\Psi_{xxxx} - \frac{h^5}{120}\Psi_{xxxxx} + O(h^6)\\
\Psi_{i,j+1} =& \Psi + h\Psi_y + \frac{h^2}{2}\Psi_{yy} + \frac{h^3}{6}\Psi_{yyy} + \frac{h^4}{24}\Psi_{yyyy} + \frac{h^5}{120}\Psi_{yyyyy} + O(h^6)\\
\Psi_{i,j-1} =& \Psi - h\Psi_y + \frac{h^2}{2}\Psi_{yy} - \frac{h^3}{6}\Psi_{yyy} + \frac{h^4}{24}\Psi_{yyyy} - \frac{h^5}{120}\Psi_{yyyyy} + O(h^6)\\
\Psi_{i+1,j+1} =& \Psi + h(\Psi_x + \Psi_y) + h^2\Psi_{xy} + \frac{h^2}{2}(\Psi_{xx} + \Psi_{yy})\\
&+ \frac{h^3}{2}\Psi_{xxy} + \frac{h^3}{2}\Psi_{xyy} + \frac{h^4}{4}\Psi_{xxyy} + O(h^5)\\
\Psi_{i-1,j-1} =& \Psi - h(\Psi_x + \Psi_y) + h^2\Psi_{xy} + \frac{h^2}{2}(\Psi_{xx} + \Psi_{yy})\\
&- \frac{h^3}{2}\Psi_{xxy} - \frac{h^3}{2}\Psi_{xyy} + \frac{h^4}{4}\Psi_{xxyy} + O(h^5)\\
\Psi_{i-1,j+1} =& \Psi - h(\Psi_x - \Psi_y) - h^2\Psi_{xy} + \frac{h^2}{2}(\Psi_{xx} + \Psi_{yy})\\
&+ \frac{h^3}{2}\Psi_{xxy} - \frac{h^3}{2}\Psi_{xyy} + \frac{h^4}{4}\Psi_{xxyy} + O(h^5)\\
\Psi_{i+1,j-1} =& \Psi + h(\Psi_x - \Psi_y) - h^2\Psi_{xy} + \frac{h^2}{2}(\Psi_{xx} + \Psi_{yy})\\
&- \frac{h^3}{2}\Psi_{xxy} + \frac{h^3}{2}\Psi_{xyy} + \frac{h^4}{4}\Psi_{xxyy} + O(h^5)
\end{aligned}$$

Inserting the expansions of Eq. (D.2) into Eq. (D.1) yields

$$\begin{aligned}\nabla^2\Psi_{i,j} =&(\beta + 2\,\gamma)(\Psi_{xx} + \Psi_{yy}) + (\alpha + 4\,\beta + 4\,\gamma)\,\frac{\Psi}{h^2} \\ &+ \beta\,\frac{h^2}{12}(\Psi_{xxxx} + \Psi_{yyyy}) + \gamma\,h^2\Psi_{xxyy} + O(h^4).\end{aligned}$$

Therefore, in order for Eq. (D.1) to be fourth-order accurate, we require that

$$\begin{aligned}\beta + 2\,\gamma &= 1, \\ \alpha + 4\,(\beta + \gamma) &= 0, \\ \beta &= 0, \\ \gamma &= 0,\end{aligned} \tag{D.3}$$

which is contradictory, and therefore no choice of α, β, and γ can be chosen to make a nine-point fourth order Laplacian. We note that if one chooses α, β, and $\gamma \neq 0$ such that first two conditions of Eq. (D.3) are fulfilled, one recovers the higher-accuracy second-order nine-point Laplacian of Eq. (3.12).

APPENDIX E

Sample Implementation of the MSD Boundary Condition

Sample Implementation of the MSD Boundary Condition

In order to demonstrate the simplicity of implementing the MSD boundary condition formulated in Chapter 4, a MATLAB implementation of the MSD boundary condition in one-dimension is shown. We assume a time-dependent complex-valued partial differential equation in the form

$$U_t = F(U),$$

where $F(U)$ can contain spatial derivatives, nonlinearities, etc. Given a MATLAB function to compute $F(U)$, an initial condition `U`, storage vector `Ut`, and using a simple forward time stepping scheme ($U_{n+1} = U_n + \mathbf{k}\, F_n$) with time-step $\mathbf{k}$, we arrive at the following code:

```
for(t=0:k:endtime)
%Store F(U) into a vector.
%Here, F(U) sets boundary values to dummy values:
   Ut = F(U);
%Apply MSD boundary condition to Ut:
   Ut(1)   = 1i*imag(Ut(2)/U(2))*U(1);
   Ut(end) = 1i*imag(Ut(end-1)/U(end-1))*U(end);
%Take step in scheme:
   U = k*Ut + U;
end
```

APPENDIX F

Summary of the Stability Results of Chapter 5

Summary of the Stability Results of Chapter 5

For the nonlinear Schrödinger equation (NLSE) defined as

$$i\,\frac{\partial \Psi}{\partial t} + a\,\nabla^2\Psi - V(\mathbf{r})\Psi + s\,|\Psi|^2\Psi = 0,$$

where $a > 0$ and s are parameters of the system and $V(\mathbf{r})$ is an external potential, the numerical stability bounds on the time-step when using the fourth-order Runge-Kutta time-stepping scheme is as follows:

In the linear case where $s = 0$ and with no external potential ($V(\mathbf{r}) = 0$), utilizing periodic, Dirichlet, or Laplacian-zero boundary conditions, the stability bound on the time-step k when using the second-order central difference (CD) scheme in a d-dimensional setting is

$$k_{\text{CD}} < \frac{h^2}{d\,\sqrt{2}\,a}, \tag{F.1}$$

while that of using a fourth-order central difference scheme (with interior points computed in the two-step high-order compact (2SHOC) methodology of Chapter 3) is

$$k_{\text{2SHOC}} < \left(\frac{3}{4}\right)\frac{h^2}{d\,\sqrt{2}\,a}. \tag{F.2}$$

The linearized stability bounds for the general NLSE are

$$k < \frac{\sqrt{8}}{\max\{\|\vec{B}\|_\infty, \|\forall L_i, L_i - \vec{G}\|_\infty\}}\,\frac{h^2}{a}, \tag{F.3}$$

where $\vec{B}$ are the boundary points as defined by Table F.1 (or in the periodic case is ignored), the elements of $\vec{L}$ is defined as

$$L_i = \frac{h^2}{a}\left(s|\Psi_i|^2 - V_i\right),$$

where the index i spans the entire grid, and $\vec{G}$ is a set of values defined in Table F.2, determined by the dimension and method being used.

Table F.1. Values of $\vec{B}$ in Eq. (F.3).

	Dirichlet ($\Psi_b = \text{const}$)	Laplacian-zero ($\nabla^2\Psi_b = 0$)	MSD ($\lvert\Psi_b\rvert^2 = \text{const}$)
B_b	0	$\frac{h^2}{a}\left(s\lvert\Psi_b\rvert^2 - V_b\right)$	$\frac{h^2}{a}\text{Im}\left[\frac{\Psi_{t,b-1}}{\Psi_{b-1}}\right]$

Table F.2. Values of $\vec{G}$ in Eq. (F.3).

Scheme $\rightarrow$	CD $O(h^2)$	2SHOC $O(h^4)$
1D	$\{4, 3, 1, 0\}$	$\frac{1}{12} \times \{64, 63, 46, 12, -3, -4\}$
2D	$\{8, 7, 6, 2, 1, 0\}$	$\frac{1}{12} \times \{128, 127, 126, 110, 109, 92, 24, 9, 8, -6, -7, -8\}$
3D	$\{12, 11, 10, 9, 3, 2, 1, 0\}$	$\frac{1}{12} \times \{192, 191, 190, 189, 174, 173, 172, 156, 155, 138, 36, 21, 20, 6, 5, 4, -9, -10, -11, -12\}$

APPENDIX G

Specifications of CPUs and GPUs Used

Specifications of CPUs and GPUs Used

Here we show the specifications of the GPU and CPU that were used for the results in this paper.

The CPU has the following relevant specifications:

```
Operating System:           MS Windows 7 Enterprise 64-bit
CPU Name:                   Intel Core i3 540
CPU Clock Speed:            3.07 Ghz
Cores:                      2
Threads:                    4
L1 Data cache:              2 x 32 KB
L1 Instruction cache:       2 x 32 KB
L2 cache:                   2 x 256 KB
L3 cache:                   4 MB
DDR3 RAM:                   4.0 GB
Memory Clock rate:          669 Mhz

Performance Information (Computed with QwikMark)
------------------------------------------------
CPU Core Performance:       61 Gflop/s
Memory Bandwidth:           6 GB/s
```

The GPU has the following relevant specifications:

```
GPU Name:                                   GeForce GTX 580 (EVGA)
```

```
CUDA Driver Version / Runtime Version:         4.1 / 4.1
CUDA Capability Major/Minor version number:    2.0
Total amount of global memory (GDDR5 RAM):     1536 MB
(16) Multiprocessors x (32) CUDA Cores/MP:     512 CUDA Cores
GPU Clock Speed:                               1.59 Ghz
Memory Clock rate:                             2025.00 Mhz
L2 Cache Size:                                 786432 bytes
Total amount of shared memory per block:       49152 bytes
Total number of registers available per block: 32768
Maximum number of threads per block:           1024
Concurrent copy and execution:                 Yes with 1 copy engine
Run time limit on kernels:                     Yes
Concurrent kernel execution:                   Yes
Device has ECC support enabled:                No

Performance Information (Computed with CUDA-Z)
----------------------------------------------
Memory Copy
  Host Pinned to Device:          596.208 MB/s
  Host Pageable to Device:        532.289 MB/s
  Device to Host Pinned:          594.033 MB/s
  Device to Host Pageable:        531.315 MB/s
GPU Core Performance
  Single-precision Float:      1622440 Mflop/s
  Double-precision Float:       204026 Mflop/s
  32-bit Integer:               814731 Mflop/s
  24-bit Integer:               813770 Mflop/s
```

APPENDIX H

NLSEmagic CUDA MEX Source Code for the 3D 2SHOC NLSE Integrator

NLSEmagic CUDA MEX Source Code for the 3D 2SHOC NLSE Integrator

```
/*-----------------------------------
NLSE3D_TAKE_STEPS_2SHOC_CUDA.cu:
Program to integrate a chunk of time-steps of the 3D Nonlinear Shrodinger Equation
i*Ut + a*(Uxx + Uyy + Uzz) - V(r)*U + s*|U|^2*U = 0
using RK4 + 2SHOC with CUDA compatable GPUs

Ronald M Caplan
Computational Science Research Center
San Diego State University

INPUT:
(U,V,s,a,h2,BC,chunk_size,k)
U  = Current solution matrix
V  = External Potential matrix
s  = Nonlinearity paramater
a  = Laplacian paramater
h2 = Spatial step-size squared (h^2)
BC = Boundary condition selection switch:  1: Dirchilet 2:MSD 3:Lap=0
chunk_size = Number of time steps to take
k  = Time step-size

OUTPUT:
U:  New solution matrix
-----------------------------------*/
#include "cuda.h"
#include "mex.h"
#include "math.h"

/*Define block size*/
const int BLOCK_SIZEX = 8;
const int BLOCK_SIZEY = 8;
const int BLOCK_SIZEZ = 6;

/*Kernel to evaluate D(Psi) using shared memory*/
__global__ void compute_D(double* Dr,    double* Di,
                          double* Utmpr, double* Utmpi,
                          double* V,     double  s,
                          double l_a,    double lh2,
                          int BC,        int L,
                          int N,         int M,  int gridDim_y)
{
    /*Declare shared memory space*/
    __shared__ double sUtmpr[BLOCK_SIZEZ+2][BLOCK_SIZEY+2][BLOCK_SIZEX+2];
    __shared__ double sUtmpi[BLOCK_SIZEZ+2][BLOCK_SIZEY+2][BLOCK_SIZEX+2];

    /*Create six indexes:  three for shared, three for global*/
    int i, j, k, blockIdx_z, blockIdx_y;
    /*Compute idx for z in cube (int division acts as floor operator here)*/
    blockIdx_z  = blockIdx.y/gridDim_y;
    /*Compute "true" idx for y in cube*/
    blockIdx_y  = blockIdx.y - blockIdx_z*gridDim_y;
    /*Now can compute j and k as if there was a 3D CUDA grid:*/
    k = blockIdx.x*blockDim.x + threadIdx.x;
    j = blockIdx_y*blockDim.y + threadIdx.y;
    i = blockIdx_z*blockDim.z + threadIdx.z;

    int sk  = threadIdx.x + 1;
```

```
    int sj  = threadIdx.y + 1;
    int si  = threadIdx.z + 1;

    int msd_ijk, msd_si, msd_sj, msd_sk;
    double A, Nb, Nb1;

    int ijk = N*M*i + M*j + k;

    if(i<L && (j<N && k<M)){
        /*Copy blocksized matrix from global memory into shared memory*/
        sUtmpr[si][sj][sk] = Utmpr[ijk];
        sUtmpi[si][sj][sk] = Utmpi[ijk];
    }

    /*Synchronize the threads in the block so that all shared cells are filled.*/
    __syncthreads();

    /*If cell is NOT on boundary...*/
    if ( (j>0 && j<N-1) && ((i>0 && i<L-1) && (k>0 && k<M-1)) )
    {
        /*Copy boundary layer of shared memory block*/
        if(si==1)
        {
            sUtmpr[0][sj][sk] = Utmpr[ijk - N*M];
            sUtmpi[0][sj][sk] = Utmpi[ijk - N*M];
        }
        if(si==blockDim.z)
        {
            sUtmpr[si+1][sj][sk] = Utmpr[ijk + N*M];
            sUtmpi[si+1][sj][sk] = Utmpi[ijk + N*M];
        }
        if(sj==1)
        {
            sUtmpr[si][0][sk] = Utmpr[ijk - M];
            sUtmpi[si][0][sk] = Utmpi[ijk - M];
        }
        if(sj==blockDim.y)
        {
            sUtmpr[si][sj+1][sk] = Utmpr[ijk + M];
            sUtmpi[si][sj+1][sk] = Utmpi[ijk + M];
        }
        if(sk==1)
        {
            sUtmpr[si][sj][0] = Utmpr[ijk - 1];
            sUtmpi[si][sj][0] = Utmpi[ijk - 1];
        }
        if(sk==blockDim.x)
        {
            sUtmpr[si][sj][sk+1] = Utmpr[ijk + 1];
            sUtmpi[si][sj][sk+1] = Utmpi[ijk + 1];
        }

        /*No synchthreads needed in this case*/

        Di[ijk] = (sUtmpi[si+1][sj][sk] - 6*sUtmpi[si][sj][sk] + sUtmpi[si-1][sj][sk] +
                   sUtmpi[si][sj+1][sk]                        + sUtmpi[si][sj-1][sk] +
                   sUtmpi[si][sj][sk+1]                        + sUtmpi[si][sj][sk-1])*1h2
                       ;

        Dr[ijk] = (sUtmpr[si+1][sj][sk] - 6*sUtmpr[si][sj][sk] + sUtmpr[si-1][sj][sk] +
                   sUtmpr[si][sj+1][sk]                        + sUtmpr[si][sj-1][sk] +
                   sUtmpr[si][sj][sk+1]                        + sUtmpr[si][sj][sk-1])*1h2
                       ;

     }/*End of interier points*/

     if(BC==2)    __syncthreads(); /* needed for MSD*/
```

```
    if(i<L && (j<N && k<M)){

        /*Cell is ON Boundery*/
        if(!( (j>0 && j<N-1) && ((i>0 && i<L-1) && (k>0 && k<M-1)) ) ){
            switch(BC){
                case 1:  /*Dirichlet*/
                    Dr[ijk]   = l_a*(V[ijk] - s*(sUtmpr[si][sj][sk]*sUtmpr[si][sj][sk] +
                        sUtmpi[si][sj][sk]*sUtmpi[si][sj][sk]))*sUtmpr[si][sj][sk];
                    Di[ijk]   = l_a*(V[ijk] - s*(sUtmpr[si][sj][sk]*sUtmpr[si][sj][sk] +
                        sUtmpi[si][sj][sk]*sUtmpi[si][sj][sk]))*sUtmpi[si][sj][sk];
                    break;
                case 2: /* Mod-Squared Dirichlet |U|^2=B */
                    if(i==0)                msd_si = si+1;
                    if(i==L-1)              msd_si = si-1;
                    if((i!=0) && (i!=L-1))  msd_si = si;
                    if(j==0)                msd_sj = sj+1;
                    if(j==N-1)              msd_sj = sj-1;
                    if((j!=0) && (j!=N-1))  msd_sj = sj;
                    if(k==0)                msd_sk = sk+1;
                    if(k==M-1)              msd_sk = sk-1;
                    if((k!=0) && (k!=M-1))  msd_sk = sk;

                    msd_ijk = N*M*(i+(msd_si-si)) + M*(j+(msd_sj-sj)) + k + (msd_sk-sk);

                    if(sUtmpr[msd_si][msd_sj][msd_sk]==0 && sUtmpi[msd_si][msd_sj][msd_sk
                        ]==0)
                    {
                        A=0;
                    }
                    else{
                    A   = (Dr[msd_ijk]*sUtmpr[msd_si][msd_sj][msd_sk] + Di[msd_ijk]*sUtmpi
                        [msd_si][msd_sj][msd_sk])/
                          (sUtmpr[msd_si][msd_sj][msd_sk]*sUtmpr[msd_si][msd_sj][msd_sk] +
                                sUtmpi[msd_si][msd_sj][msd_sk]*sUtmpi[msd_si][msd_sj][
                               msd_sk]);
                    }

                    Nb  = s*(sUtmpr[si][sj][sk]*sUtmpr[si][sj][sk] + sUtmpi[si][sj][sk]*
                        sUtmpi[si][sj][sk]) - V[ijk];
                    Nb1 = s*(sUtmpr[msd_si][msd_sj][msd_sk]*sUtmpr[msd_si][msd_sj][msd_sk]
                         + sUtmpi[msd_si][msd_sj][msd_sk]*sUtmpi[msd_si][msd_sj][msd_sk])
                        - V[msd_ijk];

                    Dr[ijk]  = (A + l_a*(Nb1-Nb))*sUtmpr[si][sj][sk];
                    Di[ijk]  = (A + l_a*(Nb1-Nb))*sUtmpi[si][sj][sk];
                    break;
                case 3: /*Lap=0*/
                    Dr[ijk] = 0.0;
                    Di[ijk] = 0.0;
                    break;
                default:
                    Dr[ijk] = 0.0;
                    Di[ijk] = 0.0;
                    break;
            }/*BC Switch*/
        }/*BC*/
    }/*end*/
}/*Compute D*/

/*Kernel to evaluate F(Psi) using shared memory*/
__global__ void compute_F(double* ktotr, double* ktoti,
                          double* Utmpr, double* Utmpi,
                          double* Uoldr, double* Uoldi,
                          double* Uoutr, double* Uouti,
                          double* Dr,    double* Di,
                          double* V,     double s,     double a,
                          double a1_6h2, double a1_12, double h2,
                          int BC,        int L,
```

```
183                             int N,        int M,  int gridDim_y, double K, int fstep)
{
185     /*Declare shared memory space*/
        __shared__ double sUtmpr[BLOCK_SIZEZ+2][BLOCK_SIZEY+2][BLOCK_SIZEX+2];
187     __shared__ double sUtmpi[BLOCK_SIZEZ+2][BLOCK_SIZEY+2][BLOCK_SIZEX+2];
        __shared__ double  NLSFr[BLOCK_SIZEZ+2][BLOCK_SIZEY+2][BLOCK_SIZEX+2];
189     __shared__ double  NLSFi[BLOCK_SIZEZ+2][BLOCK_SIZEY+2][BLOCK_SIZEX+2];
        __shared__ double    sDr[BLOCK_SIZEZ+2][BLOCK_SIZEY+2][BLOCK_SIZEX+2];
191     __shared__ double    sDi[BLOCK_SIZEZ+2][BLOCK_SIZEY+2][BLOCK_SIZEX+2];
        __shared__ double     sV[BLOCK_SIZEZ+2][BLOCK_SIZEY+2][BLOCK_SIZEX+2];
193
        /*Create six indexes:  three for shared, three for global*/
195     int i, j, k, blockIdx_z, blockIdx_y;
        /*Compute idx for z in cube (int division acts as floor operator here)*/
197     blockIdx_z  = blockIdx.y/gridDim_y;
        /*Compute "true" idx for y in cube*/
199     blockIdx_y  = blockIdx.y - blockIdx_z*gridDim_y;
        /*Now can compute j and k as if there was a 3D CUDA grid:*/
201     k = blockIdx.x*blockDim.x + threadIdx.x;
        j = blockIdx_y*blockDim.y + threadIdx.y;
203     i = blockIdx_z*blockDim.z + threadIdx.z;

205     int sk  = threadIdx.x + 1;
        int sj  = threadIdx.y + 1;
207     int si  = threadIdx.z + 1;

209     int msd_si, msd_sj, msd_sk;
        double OM;
211
        int ijk = N*M*i + M*j + k;
213
        /*Copy vector from global memory into shared memory*/
215     if(i<L && (j<N && k<M))
        {
217         sUtmpr[si][sj][sk] = Utmpr[ijk];
            sUtmpi[si][sj][sk] = Utmpi[ijk];
219            sDr[si][sj][sk] =    Dr[ijk];
               sDi[si][sj][sk] =    Di[ijk];
221             sV[si][sj][sk] =     V[ijk];

223         /*Copy boundary layer of shared memory block*/
            /*These need to be outside next if statement due to diag accesses*/
225         if(i>0 && si==1)
            {
227             sUtmpr[0][sj][sk] = Utmpr[ijk - N*M];
                sUtmpi[0][sj][sk] = Utmpi[ijk - N*M];
229             sDr[0][sj][sk]    =    Dr[ijk - N*M];
                sDi[0][sj][sk]    =    Di[ijk - N*M];
231         }
            if(i<L-1 && si==blockDim.z)
233         {
                sUtmpr[si+1][sj][sk] = Utmpr[ijk + N*M];
235             sUtmpi[si+1][sj][sk] = Utmpi[ijk + N*M];
                sDr[si+1][sj][sk]    =    Dr[ijk + N*M];
237             sDi[si+1][sj][sk]    =    Di[ijk + N*M];
            }
239         if(j>0 && sj==1)
            {
241             sUtmpr[si][0][sk] = Utmpr[ijk - M];
                sUtmpi[si][0][sk] = Utmpi[ijk - M];
243             sDr[si][0][sk]    =    Dr[ijk - M];
                sDi[si][0][sk]    =    Di[ijk - M];
245         }
            if(j<N-1 && sj==blockDim.y)
247         {
                sUtmpr[si][sj+1][sk] = Utmpr[ijk + M];
249             sUtmpi[si][sj+1][sk] = Utmpi[ijk + M];
                sDr[si][sj+1][sk]    =    Dr[ijk + M];
```

```
251             sDi[si][sj+1][sk]     =     Di[ijk + M];
                }
253             if(k>0 && sk==1)
                {
255             sUtmpr[si][sj][0] = Utmpr[ijk - 1];
                sUtmpi[si][sj][0] = Utmpi[ijk - 1];
257             sDr[si][sj][0]     =     Dr[ijk - 1];
                sDi[si][sj][0]     =     Di[ijk - 1];
259             }
                if(k<M-1 && sk==blockDim.x)
261             {
                sUtmpr[si][sj][sk+1] = Utmpr[ijk + 1];
263             sUtmpi[si][sj][sk+1] = Utmpi[ijk + 1];
                sDr[si][sj][sk+1]     =     Dr[ijk + 1];
265             sDi[si][sj][sk+1]     =     Di[ijk + 1];
                }
267         }/*end*/

269         /*Synchronize the threads in the block so that all shared cells are filled.*/
            __syncthreads();
271
            /*If cell is not on boundary...*/
273         if((j>0 && j<N-1) && ((i>0 && i<L-1) && (k>0 && k<M-1)))
            {
275             /*Copy boundary layer of shared memory block*/
                if(si==1 && sj==1)
277             {
                sUtmpr[0][0][sk] = Utmpr[ijk - N*M - M];
279             sUtmpi[0][0][sk] = Utmpi[ijk - N*M - M];
                }
281             if(si==1 && sj==blockDim.y)
                {
283             sUtmpr[0][sj+1][sk] = Utmpr[ijk - N*M + M];
                sUtmpi[0][sj+1][sk] = Utmpi[ijk - N*M + M];
285             }
                if(si==1 && sk==1)
287             {
                sUtmpr[0][sj][0] = Utmpr[ijk - N*M - 1];
289             sUtmpi[0][sj][0] = Utmpi[ijk - N*M - 1];
                }
291             if(si==1 && sk==blockDim.x)
                {
293             sUtmpr[0][sj][sk+1] = Utmpr[ijk - N*M + 1];
                sUtmpi[0][sj][sk+1] = Utmpi[ijk - N*M + 1];
295             }
                if(si==blockDim.z && sj==1)
297             {
                sUtmpr[si+1][0][sk] = Utmpr[ijk + N*M - M];
299             sUtmpi[si+1][0][sk] = Utmpi[ijk + N*M - M];
                }
301             if(si==blockDim.z && sj==blockDim.y)
                {
303             sUtmpr[si+1][sj+1][sk] = Utmpr[ijk + N*M + M];
                sUtmpi[si+1][sj+1][sk] = Utmpi[ijk + N*M + M];
305             }
                if(si==blockDim.z && sk==1)
307             {
                sUtmpr[si+1][sj][0] = Utmpr[ijk + N*M - 1];
309             sUtmpi[si+1][sj][0] = Utmpi[ijk + N*M - 1];
                }
311             if(si==blockDim.z && sk==blockDim.x)
                {
313             sUtmpr[si+1][sj][sk+1] = Utmpr[ijk + N*M + 1];
                sUtmpi[si+1][sj][sk+1] = Utmpi[ijk + N*M + 1];
315             }
                if(sj==1 && sk==1)
317             {
                sUtmpr[si][0][0] = Utmpr[ijk - M - 1];
```

```
319             sUtmpi[si][0][0] = Utmpi[ijk - M - 1];
            }
321         if(sj==1 && sk==blockDim.x)
            {
323             sUtmpr[si][0][sk+1] = Utmpr[ijk - M + 1];
                sUtmpi[si][0][sk+1] = Utmpi[ijk - M + 1];
325         }
            if(sj==blockDim.y && sk==1)
327         {
                sUtmpr[si][sj+1][0] = Utmpr[ijk + M - 1];
329             sUtmpi[si][sj+1][0] = Utmpi[ijk + M - 1];
            }
331         if(sj==blockDim.y && sk==blockDim.x)
            {
333             sUtmpr[si][sj+1][sk+1] = Utmpr[ijk + M + 1];
                sUtmpi[si][sj+1][sk+1] = Utmpi[ijk + M + 1];
335         }

337         NLSFr[si][sj][sk]      =  a1_12*(sDi[si+1][sj][sk] + sDi[si-1][sj][sk] +
                                      sDi[si][sj+1][sk] + sDi[si][sj-1][sk] +
339                                   sDi[si][sj][sk+1] + sDi[si][sj][sk-1] -
                                    10*sDi[si][sj][sk])
341          - a1_6h2*(sUtmpi[si+1][sj+1][sk] + sUtmpi[si][sj+1][sk+1] + sUtmpi[si+1][sj][sk
                 +1] +
                       sUtmpi[si-1][sj-1][sk] + sUtmpi[si][sj-1][sk-1] + sUtmpi[si-1][sj][sk
                           -1] +
343                    sUtmpi[si+1][sj-1][sk] + sUtmpi[si][sj+1][sk-1] + sUtmpi[si+1][sj][sk
                           -1] +
                       sUtmpi[si-1][sj+1][sk] + sUtmpi[si][sj-1][sk+1] + sUtmpi[si-1][sj][sk
                           +1] -
345                 12*sUtmpi[si][sj][sk])
             + (sV[si][sj][sk] - s*(sUtmpr[si][sj][sk]*sUtmpr[si][sj][sk] + sUtmpi[si][sj][sk
                 ]*sUtmpi[si][sj][sk]))*sUtmpi[si][sj][sk];
347
            NLSFi[si][sj][sk]     = -a1_12*(sDr[si+1][sj][sk] + sDr[si-1][sj][sk] +
349                                   sDr[si][sj+1][sk] + sDr[si][sj-1][sk] +
                                      sDr[si][sj][sk+1] + sDr[si][sj][sk-1] -
351                                 10*sDr[si][sj][sk])
             + a1_6h2*(sUtmpr[si+1][sj+1][sk] + sUtmpr[si][sj+1][sk+1] + sUtmpr[si+1][sj][sk
                 +1] +
353                    sUtmpr[si-1][sj-1][sk] + sUtmpr[si][sj-1][sk-1] + sUtmpr[si-1][sj][sk
                           -1] +
                       sUtmpr[si+1][sj-1][sk] + sUtmpr[si][sj+1][sk-1] + sUtmpr[si+1][sj][sk
                           -1] +
355                    sUtmpr[si-1][sj+1][sk] + sUtmpr[si][sj-1][sk+1] + sUtmpr[si-1][sj][sk
                           +1] -
                    12*sUtmpr[si][sj][sk])
357          - (sV[si][sj][sk] - s*(sUtmpr[si][sj][sk]*sUtmpr[si][sj][sk] + sUtmpi[si][sj][sk
                 ]*sUtmpi[si][sj][sk]))*sUtmpr[si][sj][sk];

359     }/*End of interier points*/

361
        if(BC==2)    __syncthreads(); /* needed for MSD*/
363
        if(i<L && (j<N && k<M)){
365
            /*Cell is ON Boundery*/
367         if(!( (j>0 && j<N-1) && ((i>0 && i<L-1) && (k>0 && k<M-1)) ) ){
                switch(BC){
369                 case 1: /*Dirichlet*/
                        NLSFr[si][sj][sk]    = 0.0;
371                     NLSFi[si][sj][sk]    = 0.0;
                        break;
373                 case 2: /* Mod-Squared Dirichlet |U|^2=B */
                        if(i==0)                msd_si = si+1;
375                     if(i==L-1)              msd_si = si-1;
                        if((i!=0) && (i!=L-1))  msd_si = si;
```

```
377                         if(j==0)                    msd_sj = sj+1;
                            if(j==N-1)                  msd_sj = sj-1;
379                         if((j!=0) && (j!=N-1))      msd_sj = sj;
                            if(k==0)                    msd_sk = sk+1;
381                         if(k==M-1)                  msd_sk = sk-1;
                            if((k!=0) && (k!=M-1))      msd_sk = sk;
383
                           if(sUtmpr[msd_si][msd_sj][msd_sk]==0 && sUtmpi[msd_si][msd_sj][msd_sk
                               ]==0)
385                         {
                                OM=0;
387                         }
                            else{
389                             OM = (NLSFi[msd_si][msd_sj][msd_sk]*sUtmpr[msd_si][msd_sj][msd_sk]
                                      - NLSFr[msd_si][msd_sj][msd_sk]*sUtmpi[msd_si][msd_sj][
                                          msd_sk])/
                                  (sUtmpr[msd_si][msd_sj][msd_sk]*sUtmpr[msd_si][msd_sj][msd_sk] +
                                       sUtmpi[msd_si][msd_sj][msd_sk]*sUtmpi[msd_si][msd_sj][msd_sk
                                           ]);
391                         }

393                         NLSFr[si][sj][sk]  = -OM*sUtmpi[si][sj][sk];
                            NLSFi[si][sj][sk]  =  OM*sUtmpr[si][sj][sk];
395                         break;
                        case 3: /*Uxx+Uyy+Uzz=0:*/
397                         NLSFr[si][sj][sk] = - (s*(sUtmpr[si][sj][sk]*sUtmpr[si][sj][sk] +
                                sUtmpi[si][sj][sk]*sUtmpi[si][sj][sk]) - sV[si][sj][sk])*sUtmpi[si
                                ][sj][sk];
                            NLSFi[si][sj][sk] =   (s*(sUtmpr[si][sj][sk]*sUtmpr[si][sj][sk] +
                                sUtmpi[si][sj][sk]*sUtmpi[si][sj][sk]) - sV[si][sj][sk])*sUtmpr[si
                                ][sj][sk];
399                         break;
                        default:
401                         NLSFr[si][sj][sk]   = 0.0;
                            NLSFi[si][sj][sk]   = 0.0;
403                         break;
                      }/*BC switch*/
405                 }/*on BC*/
                    switch(fstep)  {
407                   case 1:
                        ktotr[ijk] = NLSFr[si][sj][sk];
409                     ktoti[ijk] = NLSFi[si][sj][sk];
                        /*sUtmp is really Uold and Uold is really Utmp*/
411                     Uoldr[ijk] = sUtmpr[si][sj][sk] + K*NLSFr[si][sj][sk];
                        Uoldi[ijk] = sUtmpi[si][sj][sk] + K*NLSFi[si][sj][sk];
413                     break;
                      case 2:
415                     ktotr[ijk] = ktotr[ijk] + 2*NLSFr[si][sj][sk];
                        ktoti[ijk] = ktoti[ijk] + 2*NLSFi[si][sj][sk];
417                     Uoutr[ijk] = Uoldr[ijk] + K*NLSFr[si][sj][sk];
                        Uouti[ijk] = Uoldi[ijk] + K*NLSFi[si][sj][sk];
419                     break;
                      case 3:
421                     Uoldr[ijk] = Uoldr[ijk] + K*(ktotr[ijk] + NLSFr[si][sj][sk]);
                        Uoldi[ijk] = Uoldi[ijk] + K*(ktoti[ijk] + NLSFi[si][sj][sk]);
423                     break;
                     }/*switch step*/
425             }/*<end*/
            }/*Compute_F*/
427
            /*Main mex function*/
429         void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
            {
431             int     L,N,M,dims,gridDim_y;
                const int *dim_array;
433             int     i, c, chunk_size, BC;
                double  h2, lh2, a, l_a, a1_6h2, a1_12, s, K, K2, K6;
435             double *vUoldr, *vUoldi,  *vV, *vUnewr, *vUnewi;
```

```
437
        /*GPU variables:*/
439     double *Utmpr, *Utmpi;
        double *Uoutr, *Outi, *ktotr, *ktoti;
441     double *Uoldr_gpu, *Uoldi_gpu,  *V_gpu;
        double *Dr, *Di;
443
        /*Find the dimensions of the input cube*/
445     dims      = (int)mxGetNumberOfDimensions(prhs[0]);
        dim_array = (const int*)mxGetDimensions(prhs[0]);
447     M         = dim_array[0];
        N         = dim_array[1];
449     L         = dim_array[2];

451     /*Create output vector*/
        plhs[0] =  mxCreateNumericArray((mwSize)dims, (mwSize*)dim_array, mxDOUBLE_CLASS,
            mxCOMPLEX);
453
        /* Retrieve the input data */
455     vUoldr = mxGetPr(prhs[0]);
        if(mxIsComplex(prhs[0])){
457         vUoldi = mxGetPi(prhs[0]);
        }
459     else{
            vUoldi = (double *)mxMalloc(sizeof(double)*L*N*M);
461         for(i=0;i<L*N*M;i++){
                vUoldi[i] = 0.0;
463         }
        }
465     vV      = mxGetPr(prhs[1]);

467     /*Get the rest of the input variables*/
        s           = (double)mxGetScalar(prhs[2]);
469     a           = (double)mxGetScalar(prhs[3]);
        h2          = (double)mxGetScalar(prhs[4]);
471     BC          =   (int)mxGetScalar(prhs[5]);
        chunk_size  =   (int)mxGetScalar(prhs[6]);
473     K           = (double)mxGetScalar(prhs[7]);

475     /*Pre-compute parameter divisions*/
        l_a     = 1.0/a;
477     lh2     = 1.0/h2;
        K2      = K/2.0;
479     K6      = K/6.0;
        a1_6h2  = a*(1.0/(6.0*h2));
481     a1_12   = a*(1.0/12.0);

483     /*Allocate 1D CUDA memory*/
        cudaMalloc((void**) &Uoldr_gpu, M*N*L*sizeof(double));
485     cudaMalloc((void**) &Uoldi_gpu, M*N*L*sizeof(double));
        cudaMalloc((void**) &V_gpu,     M*N*L*sizeof(double));
487     cudaMalloc((void**) &Utmpr,     M*N*L*sizeof(double));
        cudaMalloc((void**) &Utmpi,     M*N*L*sizeof(double));
489     cudaMalloc((void**) &Dr,        M*N*L*sizeof(double));
        cudaMalloc((void**) &Di,        M*N*L*sizeof(double));
491     cudaMalloc((void**) &Uouti,     M*N*L*sizeof(double));
        cudaMalloc((void**) &Uoutr,     M*N*L*sizeof(double));
493     cudaMalloc((void**) &ktotr,     M*N*L*sizeof(double));
        cudaMalloc((void**) &ktoti,     M*N*L*sizeof(double));
495
        /*Copy input vectors to GPU*/
497     cudaMemcpy(Uoldr_gpu, vUoldr, M*N*L*sizeof(double),cudaMemcpyHostToDevice);
        cudaMemcpy(Uoldi_gpu, vUoldi, M*N*L*sizeof(double),cudaMemcpyHostToDevice);
499     cudaMemcpy(V_gpu,     vV,     M*N*L*sizeof(double),cudaMemcpyHostToDevice);

501     /*Set up CUDA grid and block size*/
        dim3 dimBlock(BLOCK_SIZEX,BLOCK_SIZEY,BLOCK_SIZEZ);
```

```
503
        /*Compute desired y grid dimension*/
505     gridDim_y  = (int)ceil((N+0.0)/dimBlock.y);

507     /*For 3D need to extend y-grid dimention to include z-cuts:*/
        dim3 dimGrid((int)ceil((M+0.0)/dimBlock.x), gridDim_y*((int)ceil((L+0.0)/dimBlock.z)))
            ;
509
        /*Compute chunk of time steps*/
511     for (c=0; c<chunk_size; c++)
        {
513       compute_D <<<dimGrid,dimBlock>>>(Dr,Di,Uoldr_gpu,Uoldi_gpu,V_gpu,s,l_a,lh2,BC,L,N,M,
              gridDim_y);
          compute_F <<<dimGrid,dimBlock>>>(ktotr,ktoti,Uoldr_gpu,Uoldi_gpu,Utmpr,    Utmpi,
                V_gpu,V_gpu,Dr,Di,V_gpu,s,a,a1_6h2,a1_12,h2,BC,L,N,M,gridDim_y,K2,1);
515       compute_D <<<dimGrid,dimBlock>>>(Dr,Di,Utmpr,Utmpi,          V_gpu,s,l_a,lh2,BC,L,N,M,
              gridDim_y);
          compute_F <<<dimGrid,dimBlock>>>(ktotr,ktoti,Utmpr,    Utmpi,    Uoldr_gpu,Uoldi_gpu
              ,Uoutr,Uouti,Dr,Di,V_gpu,s,a,a1_6h2,a1_12,h2,BC,L,N,M,gridDim_y,K2,2);
517       compute_D <<<dimGrid,dimBlock>>>(Dr,Di,Uoutr,Uouti,          V_gpu,s,l_a,lh2,BC,L,N,M,
              gridDim_y);
          compute_F <<<dimGrid,dimBlock>>>(ktotr,ktoti,Uoutr,    Uouti,    Uoldr_gpu,Uoldi_gpu
              ,Utmpr,Utmpi,Dr,Di,V_gpu,s,a,a1_6h2,a1_12,h2,BC,L,N,M,gridDim_y,K,  2);
519       compute_D <<<dimGrid,dimBlock>>>(Dr,Di,Utmpr,Utmpi,          V_gpu,s,l_a,lh2,BC,L,N,M,
              gridDim_y);
          compute_F <<<dimGrid,dimBlock>>>(ktotr,ktoti,Utmpr,    Utmpi,    Uoldr_gpu,Uoldi_gpu
              ,V_gpu,V_gpu,Dr,Di,V_gpu,s,a,a1_6h2,a1_12,h2,BC,L,N,M,gridDim_y,K6,3);
521     }

523     /*Set up output vectors*/
        vUnewr = mxGetPr(plhs[0]);
525     vUnewi = mxGetPi(plhs[0]);

527     /*Make sure everything is done (important for large chunk-size computations)*/
        cudaDeviceSynchronize();
529
        /*Transfer solution back to CPU*/
531     cudaMemcpy(vUnewr,Uoldr_gpu, M*N*L*sizeof(double),cudaMemcpyDeviceToHost);
        cudaMemcpy(vUnewi,Uoldi_gpu, M*N*L*sizeof(double),cudaMemcpyDeviceToHost);
533
        /*Free up GPU memory*/
535     cudaFree(Uoutr);
        cudaFree(Uouti);
537     cudaFree(ktotr);
        cudaFree(ktoti);
539     cudaFree(V_gpu);
        cudaFree(Uoldr_gpu);
541     cudaFree(Uoldi_gpu);
        cudaFree(Utmpr);
543     cudaFree(Utmpi);
        cudaFree(Dr);
545     cudaFree(Di);

547     if(!mxIsComplex(prhs[0])){
            mxFree(vUoldi);
549     }
    }
```

APPENDIX I
Setup Guide for Compiling CUDA MEX Codes

Setup Guide for Compiling CUDA MEX Codes

The purpose of this guide is to allow you to set up CUDA with MATLAB without having to spend days sifting through forums trying to find a path entry or link. This document will not cover good CUDA programming, but rather just how to set up CUDA, set up `nvmex`, and be able to compile CUDA programs in MATLAB.

As of MATLAB 2010b, CUDA support is built into MATLAB. This is good and bad. On the good side, it allows you to use your GPU simply within MATLAB. On the bad side, code is often faster when compiled using your own CUDA/C codes for use with MATLAB. The new MATLAB lets you do this in a roundabout way by creating GPU variables and passing them to your kernels. However, many people would rather leave MATLAB out of it and instead use a pure C-code with CUDA compiled in a MATLAB compatible MEX file. This ensures better portability as well. This guide is a "how-to" to do this. The setup info here has been tested on 32-bit Windows 7/Vista for MATLAB 2008b, and 64-bit Windows 7 with MATLAB 2010a.

Most of the information in the guide is taken from NVIDIA's manuals and various forums.

Website for this guide with all setup files
`http://www.nlsemagic.com/files/cudamatlab.zip`

1) Buy your CUDA card

The first step to setting up CUDA is to buy and install a CUDA-enabled GPU card. If you are interested in just developing codes and not running huge computations, I suggest you search Amazon for the lower-end cards that support CUDA, in which case you can find cards for $50. If you want to run serious codes, then you will want to spend more money (the more money, the more speedup). For scientific computing, you also definitely want to get a card with Fermi architecture (400 series and above).

For a list of CUDA-enabled cards go to:
`http://developer.nvidia.com/cuda-gpus`

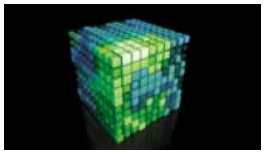
Caution! Most CUDA cards require a PCI-Express slot and a large power supply on your PC. If you have an older PC, you may only be able to use the older cards unless you buy a new power supply.

2) Download and Install Most Recent Drivers

Once your card is installed, windows will automatically install a driver for it.

Even so, it is best to go to `http://www.nvidia.com/drivers` to get the newest drivers.

3) Download and Install CUDA Toolkit



Now you need to install the CUDA Toolkit by going to `http://developer.nvidia.com/cuda-downloads` and downloading the newest version. (As of this writing, the newest toolkit is version 4.1).

Download the toolkit and any other items you want (but not the developers drivers).

There should not be much of a problem here, as these are self-extracting/installing exe files.

4) Get MATLAB

Most likely you already have MATLAB, but if not, go get it and install it.

`http://www.mathworks.com/products/matlab/`

If you are a student, the student edition is all you need and it costs $99.

I would suggest buying it from your University bookstore because it makes the student-authentication process during installation easier.

5) Get and Install Visual C++ Express 2008/2010

In order for the `nvmex` compiler to work with MATLABs MEX compiler, it needs to use the C compiler that comes with Visual C++ (others have been mentioned to work, but I have not tested them). Microsoft provides a free version of all the Visual Studios called Express. The newest full version of Visual C++ Express is 2010, but older versions of MATLAB MEX do not support it. Therefore the best plan is to get the older 2008 version of Visual C++ Express.

To get Visual C++ Express 2010, go to:

`http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express`

To get Visual C++ Express 2008, go to:

`http://www.microsoft.com/visualstudio/en-us/products/2008-editions/express`

Simply run the downloaded exe file to install it and then restart your computer.

6) Get and Install Microsoft Windows SDK

Usually the Microsoft SDK is already installed on the system, but many times (especially for 64-bit) it is not.

To install the SDK, go to
`http://www.microsoft.com/download/en/details.aspx?id=8279`
and download the SDK and run the installer. If this link fails, simply search microsoft.com for the Windows SDK.

7) Get NVMEX Compiler Package and Install

Next you need the CUDA enabled MEX compiler called `nvmex`. This is included in the Legacy MATLAB plug-in on the NVIDIA site (see `http://developer.nvidia.com/matlab-cuda`). However, NVIDIA stopped updating the plugin as soon as MATLAB integrated GPU support in their newer versions. Therefore, the online user community has taken over and after combing through forums on multiple sites, I have collected an up-to-date `nvmex` package. The new package is geared for MATLAB 2008a and above, so if your MATLAB is older than that, you should still use the updated plugin bat files, but replace the other files with those from the NVIDIA plug-in site.

To download the most current and up-to-date nvmex package, go to `http://www.nlsemagic.com/files/cudamatlab.zip`.

To install the files, first rename the appropriate nvmex.pl file (32-bit or 64-bit) to "nvmex.pl" and put it in the MATLAB bin folder, for example:
`C:\Program Files\MATLAB\R2010a\bin\`

The other important files are **nvmex.m**, **nvmex_helper.m**, and **nvmexopts.bat**. These files must be in the directory that you want to compile CUDA MEX files. There are two **nvmexopts.bat** files provided, one for 32-bit and one for 64-bit (**nvmexopts64.bat**).

For 64-bit Windows, an additional included file is necessary: **vcvarsamd64.bat**.
This file must be placed in the directory (or equivalent):
`C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\bin\amd64\`

8) Edit nvmexopts(64).bat PATHs

This is the most annoying part of the installation. I have tried to make it as easy as possible by arranging the bat files to show which lines need to be altered as follows:

In the 32-bit bat file:

```
rem *****These two lines are all that should need to be tweaked (and
target GPU below):
rem *****(Make sure you have installed the MS SDK and VS express 8)

set VSINSTALLDIR=C:\Program Files\Microsoft Visual Studio 9.0
set SDKDIR=C:\Program Files\Microsoft SDKs\Windows\v6.0A
```

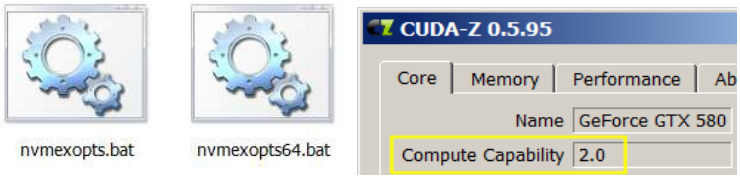
In the 64-bit bat file:

```
rem *****These two lines are all that should need to be tweaked (and
target GPU below):
rem *****(Make sure you have installed the MS SDK and VS express 8)

set VSINSTALLDIR=C:\Program Files (x86)\Microsoft Visual Studio 9.0
set SDKDIR=C:\Program Files\Microsoft SDKs\Windows\v7.0
```

These directories need to be set to the correct path. As of this writing, the paths above should work.

9) Set GPU Compute Capability in nvmexopts(64).bat



In order to use double precision with cards that support it, and to get faster codes in general, it is a good idea to compile the codes specifically for the GPU version you have installed. First, find out what compute capability your card has (either look it up online, download and run CUDA-Z, or run the CUDA SDK example "deviceQuery.exe").

Next, add to the compile options in the .bat file as follows:

set COMPFLAGS=-gencode=arch=compute_20,core=sm_20
-gencode=arch=compute_20, code=compute_20 ***(rest of original line here)***

The **code=sm_xx** generates CUBIN files that will only work on your specific compute capability, while the **code=compute_xx** generates PTX files that are compiled at run time and are forward-compatible within the same major compute capability. The codes will run a bit faster if you include native CUBIN, but are more portable if you use PTX, so I use both. The _xx can be _12, _13, _20, etc. where the number is the compute capability (i.e.

_20 is for compute capability 2.0 etc.

In order to use double precision in your code, you must have a card with compute capability 1.3 or above, and compile the codes with at least _13.

10) Compile CUDA Code from MATLAB

```
>> nvmex -f nvmexopts64.bat getCudaInfo.cu -IC:\cuda\include -LC:\cuda\lib\x64 -lcudart -lcuda
getCudaInfo.cu
tmpxft_000011fc_00000000-3_getCudaInfo.cudafe1.gpu
tmpxft_000011fc_00000000-8_getCudaInfo.cudafe2.gpu
getCudaInfo.cu
tmpxft_000011fc_00000000-3_getCudaInfo.cudafe1.cpp
>> getCudaInfo
```

Now you want to make sure everything is working.

The command in MATLAB to compile your .cu CUDA code is typically:

```
nvmex -f nvmexopts.bat yourcudacode.cu -IC:\cuda\include -LC:\cuda\lib
-lcufft lcudart lcuda
```

On 64-bit Windows, you must use

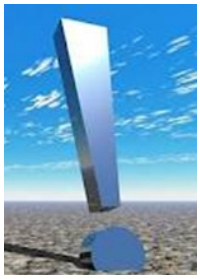
```
nvmex -f nvmexopts64.bat yourcudacode.cu -IC:\cuda\include
-LC:\cuda\lib\x64 -lcufft -lcudart -lcuda
```

The newest NVIDIA SDK does NOT install the CUDA library and include files into `C:\CUDA\` like it used to. Therefore one must use the full path in the compiler line above. On Windows, this presents a difficulty since there are often spaces in the directory names, which the compiler does not like.

The way around the problem (and what I do) is to create a folder called CUDA on the C drive (`C:\CUDA\`) and copy the folders **/lib/x64** (or **/lib/Win32**) and **/include** from the CUDA Toolkit directory (for example, **C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v4.1**) into **C:\CUDA**. If you do this, the compile lines as shown above should work fine.

Important! Don't forget to update the files in your `C:\CUDA\` directory when updating the CUDA toolkit!

That concludes the basic setup!



ONE LAST THING: When running CUDA MEX codes through MATLAB, occasionally with heavy use, crashes, NANs, programming errors, etc, MATLAB can cause a problem where the CUDA codes stop working and return garbage or crash MATLAB. This is a problem with MEX not CUDA so newer versions of MATLAB may not have this issue. If this starts happening, simply close MATLAB and restart it and everything should work fine (assuming your CUDA code has no bugs!). If that does not work, shut down the machine, unplug it, wait 30 seconds, replug it, and reboot to try again (doing this clears the RAM, which can solve a lot of problems).

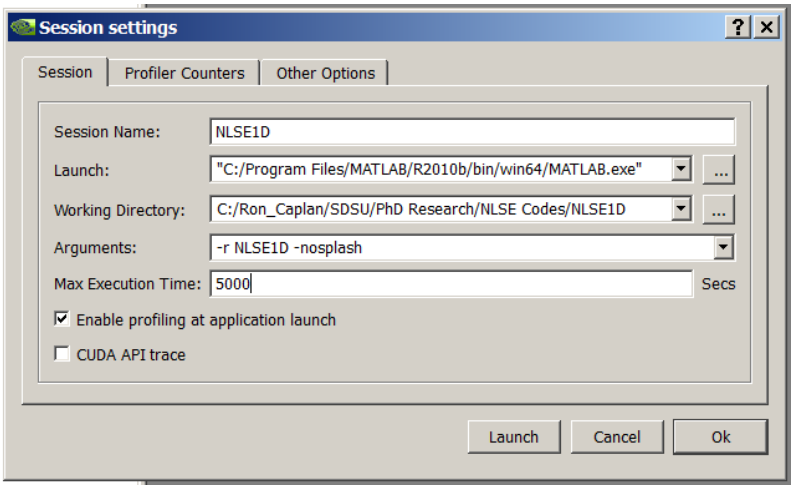
ADDITIONAL SETUP: The following are additional tips that are very useful when running CUDA codes with MATLAB on Windows:

CUDA Visual Profiler Setup

To use the profiler, you must set up the profiler to run the MATLAB executable. It is vital to NOT point to the **/MATLAB/matlab.exe**, but rather to the specific **/MATLAB/win32/matlab.exe** (or **/MATLAB/win64/matlab.exe,** or, on new versions of MATLAB, **/MATLAB/bin/win64/matlab.exe**).

Also, be sure to put an "exit;" command at the end of your MATLAB script, since the profiler will run your code multiple times and you do not want a bunch of MATLAB windows eating up RAM and desktop space.

Therefore, the setup window for visual profiler should look like this:



In order to profile your CUDA code, you will want to use the CUDA Visual Profiler.

A couple important things to note:
1) Notice that the working directory is where my MATLAB code is, not the folder where the MATLAB exe is located.
2) I use the -r tag to run the specific MATLAB script file I want, and the -nosplash to not have to see the MATLAB logo display each run.
3) I set the max execution time to 5000 so that there will definitely be enough time to finish the simulation.

For information on using the profiler results, see the CUDA documentation.

Setting up a GeForce/Quadro GPU Card as a Compute-Only Device

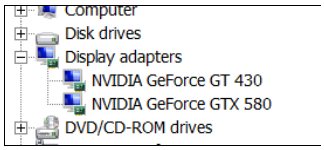
One drawback (besides the 25% double-precision performance and no ECC support) to using a GeForce or Quadro card for computing is that it is also typically being used for running the graphics of the OS which limits its capabilities to run code. Also, it makes the system pretty much unusable while the CUDA codes are running. An easy solution is to buy a Tesla model compute-only card, but those cost about 4 or 5 times as much as a GeForce.

To fix this problem in a cheaper way, you can set up two GPUs on the same PC, one being used to run the display, and the other only used for CUDA computations. I have only done this on 64-bit Windows 7 so I do not know if it works on other setups.

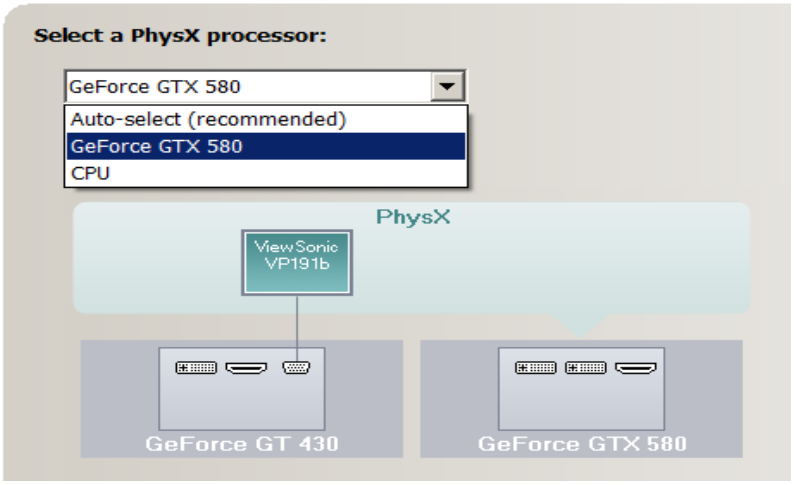
Assuming you have a motherboard with two PCI-E slots, you can set up a GeForce card as a compute-only device as follows:

First, you will need a second (probably cheaper unless you are a gamer) NVIDIA GPU card, which should be set up on the primary PCI-E slot. Then, run the system and install

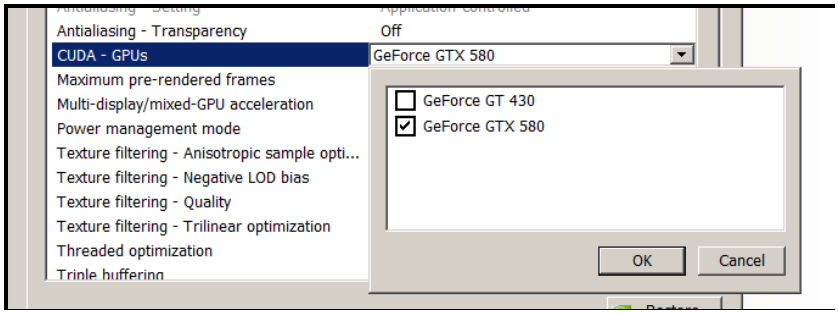
the drivers for this card. Next, shut down and install the GPU you want to compute with on the second slot, and run the system and install its drivers. If all worked out, you should see two NVIDIA cards listed in the Windows device manager like this:



Once this is done, go to the PhysX setup in the NVIDIA control panel. Select the option to set the compute GPU as the only PhysX device (I do not know if this is absolutely necessary but I did it and it works so I am including this step).



Now, if the compute-only GPU is better than the primary it should automatically be set to be the CUDA device 0 so all your CUDA codes will run on it by default without having to add device selection code. To ensure this is the case, you can manually disable the CUDA capability of the GPU being used for the display by going to the 3D options in the NVIDIA control panel and un-checking the display GPU from the CUDA selection boxes as shown here:



Important! These settings automatically get reset when installing a new driver. Therefore, be sure to change these settings back after applying a driver update.

APPENDIX J
NLSEmagic Installation Guide

NLSEmagic Installation Guide

Pre-compiled Binaries Installation

1. On Windows, make sure that the Visual Studios real-time libraries are installed on the system. The best way to do this is to install the free Visual C++ Express.

2. Unzip all files into the directory of your choice.

3. For running the CUDA integrators, be sure to have the most recent drivers installed for your GPU.
   Also, place the `nvmex.pl` file into your MATLAB bin directory (this is not needed to run the code but its presence in that directory is how the NLSEmagic driver scripts currently detects if CUDA is available or not).

4. Run MATLAB and navigate to the directory where you unzipped the files.

5. Run **NLSEmagicxD.m** (x=1,2 or 3)

Source Code Installation

1. Follow instructions 1- 4 above.

2. On Windows, run “mex –setup” and choose the Visual C++ as the compiler.

3. On Linux, modify the “makefile” to point to proper paths.

4. Run the script **makeNLSEmagicxD.m** (x=1,2, or 3)

5. In order for the GPU codes to compile, you must follow all of the instructions given in the “Setup Guide for Compiling CUDA MEX Codes” included in the NLSEmagic paekage.

6. Run **NLSEmagicxD.m** (x=1,2 or 3)