

A Customizable MATLAB GUI-Based Program for Integrating Explicit Finite Difference Schemes for the Three-Dimensional Nonlinear Schrödinger Equation and Visualizing Results

R. M. Caplan

May 6, 2009

1 Introduction

There has and continues to be much research in systems governed by variant forms of the Nonlinear Schrödinger Equation (NLS) [1]. When studying such systems numerically in three dimensions, the implementation of numerical methods can be complex. And since, typically, each researcher has to write their own code to simulate the NLS and analyze the results, this complexity can result in a lot of extra time and energy which could otherwise have been devoted to the problem being studied.

Our goal in this report is to introduce a 3D integrator for the NLS which can be customized for specific research needs. The code is made freely available online so that others can update it for their own specific problems and/or to update its numerical methodologies. As more people add to the code, it becomes a more useful research tool for everyone, as well as an educational tool for instructors.

The program is written in MATLAB using version Student Version 7.7.0 Release 2008b. MATLAB was chosen because of its ease of use, and integrated visualization features. Also, by using vector operations, and good programming practices, the run time of the MATLAB code is very comparable to codes written in C or FORTRAN.

The program in its initial form comes with many features built in, such as visualizations, saving moves, images, simulation parameters, etc. In this report we introduce our program and explain how to use it and add to it.

2 The Nonlinear Schrödinger Equation

The nondimensionalized NLS is given by:

$$i\Psi_t + \nabla^2\Psi + V(\mathbf{r})\Psi + N(|\Psi|^2)\Psi = \mathbf{0}, \quad (1)$$

where $N(|\Psi|^2)$ describes the nonlinearity, $V(\mathbf{r})$ is the potential function, and $\mathbf{r} = \{\mathbf{x}, \mathbf{y}, \mathbf{z}\}$. The nonlinearity and potential function vary per application. Some common nonlinearity functions are:

$$N(|\Psi|^2) = s_1|\Psi|^2, \quad (2)$$

$$N(|\Psi|^2) = s_1|\Psi|^2 + s_2|\Psi|^4, \quad (3)$$

$$N(|\Psi|^2) = s_1 \frac{|\Psi|^2}{1 + |\Psi|^2}, \quad (4)$$

$$(5)$$

where $s_i = \pm 1$. Eq. (2) is the standard cubic nonlinearity used in many applications including simulating Bose-Einstein condensates and nonlinear optics [2]. Eq. (3) is called the cubic-quintic nonlinearity which is

used mainly for optics as it is the first two Taylor expansion terms from the saturable nonlinearity of Eq. (4) [3]. The values of s_i determine if the nonlinearity is ‘focusing/attractive’ versus ‘defocusing/repulsive’ or for the cubic-quintic case, a competition between the two.

The potential function $V(\mathbf{r})$ can take many forms as well (such as a parabolic trapping potential or a lattice structures), and in some applications is zero-valued.

3 Finite Difference Integrator

For integrating the NLS, we have chosen to use only a select subset of possible numerical methods. The common factor in all our chosen methods is that they are all explicit finite difference (FD) schemes. Although this means we cannot guarantee unconditional stability, it has the computational advantage of not having to do a full matrix solve (along with some kind of iteration scheme for the nonlinearity) for every time step.

In order to give an option to check results with various schemes for consistency, we provide several different FD schemes.

We write the NLS as:

$$\Psi_t = F(\Psi) = i[\nabla^2\Psi + V\Psi + N(|\Psi|^2)\Psi],$$

which allows us to separate our treatment of the spacial derivatives and the time integration. Although many schemes use clever combinations of the the time and space derivatives [4], for now, all our schemes treat them separately.

We will denote the solution at time $t = kn$ and position $x = h_x l$, $y = h_y m$, $z = h_z q$ as:

$$\Psi(x, y, z, t) \equiv U_{l,m,q}^n,$$

where $k = \Delta t$, $h_x = \Delta x$, $h_y = \Delta y$, and $h_z = \Delta z$.

3.1 Time Methods

Here we describe our different options for the time integration.

3.1.1 Forward Time - $O(k)$

Our first option for the time integration is the standard first-order forward time (FT) difference given by:

$$U^{n+1} = U^n + k F(U^n). \tag{6}$$

This scheme is not very useful, as it requires very small time steps to be stable, but it is the simplest to implement and so it is useful for checking the results against the other schemes.

3.1.2 Leap Frog (Central Time) - $O(k^2)$

The next option we give is the leap-frog (LF) scheme given by:

$$U^{n+1} = U^{n-1} + 2k F(U^n). \tag{7}$$

This scheme is a two-step method, which requires an initialization one-step differencing for which we use the FT. This scheme allows a greater time step to be used, and is still very simple. For some PDEs, the LF is unconditionally stable [4], but it does not appear that this is the case for the NLS.

3.1.3 Runge-Kutta - $O(k^4)$

The last option we give (currently) is the classic 4th-order Runge-Kutta (RK4) scheme given by [5]:

$$U^{n+1} = U^n + \frac{k}{6} (q_1 + 2q_2 + 2q_3 + q_4), \quad (8)$$

where

$$\begin{aligned} q_1 &= F(U^n), & q_2 &= F(U^n + \frac{k}{2}q_1), \\ q_3 &= F(U^n + \frac{k}{2}q_2), & q_4 &= F(U^n + kq_3). \end{aligned}$$

This scheme is very accurate but computationally expensive. However, the allowed increase in k more than compensates for the increased complexity.

3.2 Spatial Methods

For computing the spatial derivatives, we do not limit ourselves to equal spacing in x , y , and z . Thus our grid is not necessarily uniform (although it is uniform along any specific direction).

3.2.1 Central Difference - $O(h^2)$

Currently, we only provide one option for computing the Laplacian, which is the standard 7-point second order central differencing, given by:

$$\nabla^2 U = \frac{U_{l-1,m,q} - 2U_{l,m,q} + U_{l+1,m,q}}{h_x^2} + \frac{U_{l,m-1,q} - 2U_{l,m,q} + U_{l,m+1,q}}{h_y^2} + \frac{U_{l,m,q-1} - 2U_{l,m,q} + U_{l,m,q+1}}{h_z^2}. \quad (9)$$

We are currently looking into the possibility of developing a 4th-order 15-point High-Order Compact (HOC) Laplacian if possible, and perhaps even a 6th-order 27-point HOC Laplacian. Another option that may be looked into is using Mimetic methods which have very good conservation properties [6].

3.3 Boundary Conditions

Here we describe the boundary condition options available. These are very critical, because having the wrong boundary conditions can make an otherwise stable scheme go unstable in unpredictable ways [4].

3.3.1 Dirichlet

One kind of BC is to simply set the value of the wave function at the boundary to a constant value:

$$\partial\Psi = C. \quad (10)$$

Typically, one uses this condition for problems whose initial conditions are far from the boundary and decay to the constant C as they approach the boundary. Often this constant takes the value 0 (for example when simulating localized bright objects, or when the potential function decays the wave function towards the boundaries).

In our program we give two options of this kind. One is to set the boundary value to 0, and the other is to set the boundary to the initial condition's maximum value. This is useful for simulating defocusing 'dark' objects in a constant, infinite background.

3.3.2 Extrapolations

Next, we implement a simple extrapolation BC. We set the value at the boundary equal to the value one grid point into the cube. This creates an inner boundary in which the Laplacian can be computed using the extrapolated boundary points. This only works well if U is not varying too much near the boundaries.

Further kinds of extrapolation could be added, such as using a quadratic or cubic extrapolation using interior grid points.

3.3.3 Periodic

Another BC is to have a periodic BC where the Laplacian is altered so that points over the edge of the computational space are taken from the opposite side of the data cube. This can be very useful for studying certain problems, such as the stability of moving structures, or simulating infinite background problems. An example is shown in Fig. 1.

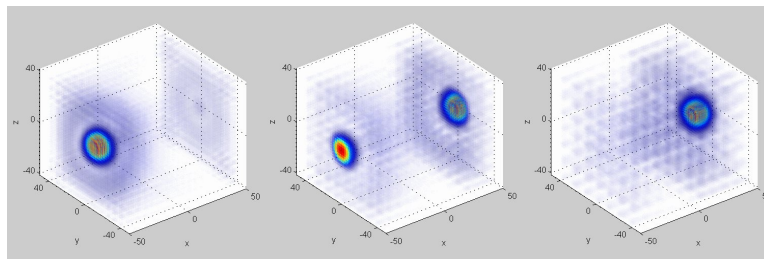


Figure 1: Example of periodic boundary conditions. Shown it the mod-squared of a moving cubic-quintic light bullet going through one side of the grid and appearing out the other side.

3.3.4 Periodic - Infinite Tube / 2D Mode

We also give the option to have Dirichlet zero-value conditions in Y and Z combined with periodic conditions in the X direction alone. This kind of BC is good for simulating problems such as infinite-length objects.

Also, if one sets the X range to go from 0 to 1 with a step length of 1, then this BC essentially turns the code into a 2D simulator (albeit with some redundant computation in the Laplacian).

3.4 Initial Conditions

The initial condition for different problems will of course be very diverse. The initial condition generator in the file `rmc_NLS3D_ICG.m` gives one access to the x , y , and z grid points to use to form an initial condition.

We include some initial conditions from our own research interests as examples. One example of interest is the vortex tube IC in which we solve for a steady-state cubic-quintic bright vortex in a 1D polar grid using a nonlinear optimization routine [7]. We then interpolate the resulting 2D polar IC into the 3D Cartesian grid as a vortex tube. The code included in this example may be of use to others who want to interpolate polar ICs into the code.

4 The Program Interface

Here we describe the interface to run the program. The program can be run either using the GUI, or by the command line. To run with the GUI, simply type ‘`rmc_NLS3D_GUI`’ at the MATLAB command line.

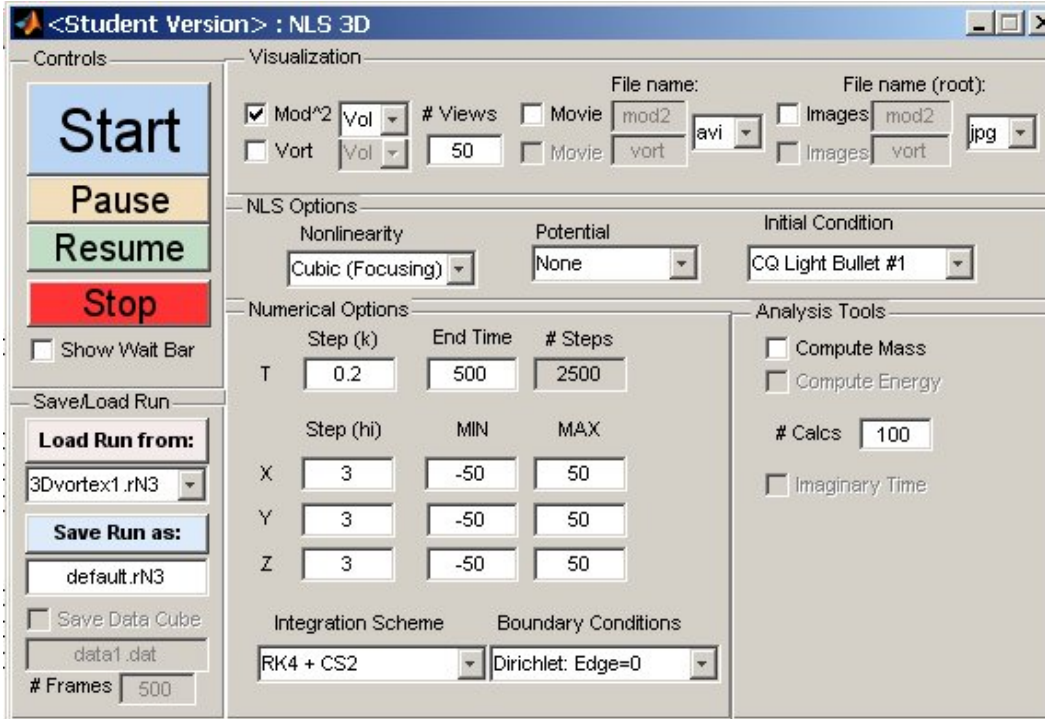


Figure 2: GUI for rmc_NLS3D.

4.1 The GUI

The main GUI is shown in Fig. 2.

The start button begins the simulation by displaying the initial condition and then the program pauses. Simply press any key to start the run. The pause and resume buttons allow one to pause the run anytime in order to view 2D slices, manipulate the 3D visualization view angle, zoom, or save images from the figure windows. Afterwards, the resume button will continue the simulation. The stop button stops the run but does so in a clean way (i.e. closes and saves movie files, closes the wait bar, returns the MATLAB command line to the user, etc.). This is a much preferable way to stop a simulation prematurely than using Control-C.

Below the controls are the save/load feature. Once all the parameters in the GUI are set to the desired setting, one can save the configuration by pressing the save run button. It will save the run with the filename specified by the text box below the button. To load a run, simply select the file you want to load from the pull down menu (which automatically finds all saved runs, and is automatically updated when a new run is saved) and press the load run button. All saved runs must be in a folder called 'SavedRuns', which must exist in the directory the code is stored for the program to work correctly. Also, it should be noted that the way the program saves a run is by saving all the parameters in a structure object. When a run is loaded, these parameters are copied over in a subroutine in `rmc_NLS3D_GUI.m` called `'load_get_params()'`. If one adds more features to the GUI (such as new analysis options, etc.), one must alter this routine to load the new parameters. This will not cause problems loading older saved runs, since existence of each parameter is tested before loading (the new feature's value will just be the default).

The NLS options are self explanatory. However, to add a new initial condition, one must run guide by typing `'guide'` in MATLAB and then double clicking on the initial condition drop list, add the new IC to the string variable, and then test for this new condition in the file `rmc_NLS3D_ICG.m`.

The numerical options let you choose the max and min values of the grid, the grid spacing, the time step, the total time, etc. Also the drop boxes at the bottom let one choose which integration scheme to use, as well as which boundary conditions to use. To add new schemes and BCs, simply follow the instructions given

for the ICs. (Note, however that there are two BC files, one for Laplacian boundaries (`rmc_NLS3D_BC_L.m`) and one for boundary conditions on the wave function itself (`rmc_NLS3D_BC_W.m`)).

The analysis tools panel currently only lets you computing the ‘mass’ defined by:

$$M = \int_{-\infty}^{\infty} |\Psi|^2 d\mathbf{r},$$

which can be helpful in testing if schemes/BCs are conservative. This is implemented only as an example. Typically, the analysis panel will consist of research-specific needs.

The visualization panel will be discussed later.

4.2 Non-GUI Mode

The program does not need the GUI to run. In fact, all the GUI does is set all the options and parameters and then calls the main program in the home work space of MATLAB. To run without the GUI just type `rmc_NLS3D` at the MATLAB prompt. To change options, ICs, etc. one must edit the file `rmc_NLS3D_SETPARAMS.m` with the options desired. The only features that running the code without the GUI lack are the ability to pause and resume the run, as well as the save and load run feature.

5 Visualization

Here we describe the visualizations in the program.

5.1 Visualization Data

When visualizing results, one often is interested in different kind of data depending on ones research needs. A most common quantity that researchers using the NLS are interested in is the mod-squared ($|\Psi|^2$) of the wave function which represents the observable phenomenon (i.e. the density in BECs, intensity in optics).

For our current research, we are studying vortex structures, in which case a quantity called ‘vorticity’ is of interest, defined as [8]:

$$\vec{\omega} = \nabla \times \frac{\Psi^* \nabla \Psi - \Psi \nabla \Psi^*}{i|\Psi|^2}. \quad (11)$$

In our program, we therefore have an option to visualize the 2-norm of the vorticity.

Additional visualization data could be added to the code as well, by mimicking the code/gui modules in the visualization panel.

5.2 Visual Modes

Currently, only one form of 3D visualization (for each visual data type) is implemented in `rmc_NLS3D`, that of volumetric rendering. This is achieved by invoking a code written by Joe Conti called ‘`vol3D.m`’. As of this writing, the transparency map (called the alpha map) is set for bright objects. This alpha map needs to be changed for dark objects to be visible.

An example volumetric rendering of the mod-squared of a near-collision of two cubic-quintic ‘light-bullets’ (stable soliton-like blobs) is given in Fig. 3.

5.3 Visualizing Slices

When the user hits the ‘pause’ button, a slice visualization is automatically initiated in a separate window for the mod-squared. The slice plots are from a free code called ‘SliceBrowser’ by Marian Uhercik. With this code, one can scan through the data cube using 2D slice plots. Afterwards, one can close the slice window and press ‘resume’ to continue the simulation. The code can be easily altered to show slice plots of the vorticity, phase, etc. by altering the callback for the pause button in `rmc_NLS3D_GUI.m`. An example is shown in Fig. 4.

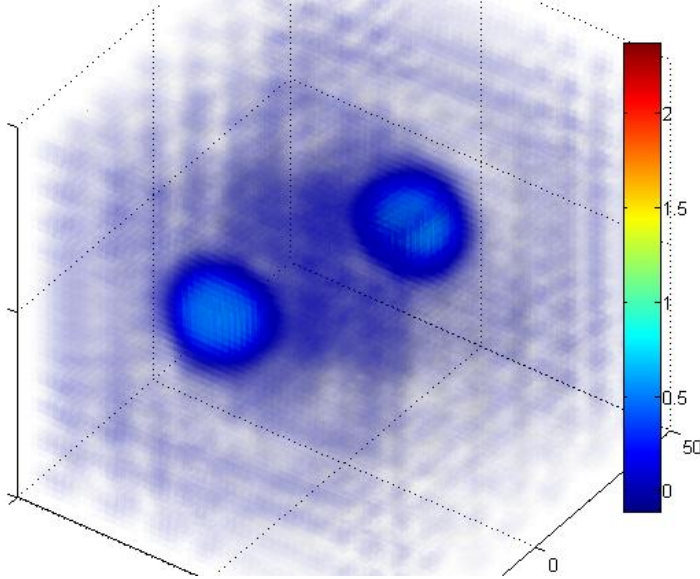


Figure 3: Example of 3D volumetric plot of the mod-squared of 2 cubic-quintic light-bullets after a near collision.

5.4 Saving Movies and Images

Saving movies and images of simulations is currently the only way to save results. The movie and image saving controls are in the visualization panel on the GUI. The number of views determines how many equally-time-spaced frames to display and/or save in the movie or numbered images.

There are various file formats available for both movies and sequential images. For movies, one can save them in either avi or animated gif format. The animated gifs are very useful for posting movies on a web site. For images, one can save them as jpg, gif, pdf, or eps.

6 Example Use

As a quick example of using the program, we show an example from our research.

In our current research, we are studying the azimuthal modulational instability (MI) of vortices in the cubic-quintic NLS. The problem is two-dimensional. To test for stability, we can integrate a vortex solution for a long time and see if it breaks up into filaments. Our goal is to test for the critical value of the complex frequency, which is where the vortices become stable. To do this we need to run many simulations around the predicted values of the critical frequency and view the results. An example run of this kind (for the unstable case) is shown in Fig. 5. We have used the x-only periodic boundary conditions and set the x-range to $[0, 1]$ with $h_x = 1$, essentially using the code in 2D mode. The vortex is produced in the initial condition generator and interpolated to the Cartesian grid. For more information, see our study for the cubic NLS vortices in Ref. [2].

7 Possible Extensions

Besides the possible extensions to the program mentioned throughout this report, we mention briefly some other ideas for improvements.

First, it would be ideal to have the 3D visualizations be a separate GUI, so that more visual options can be displayed. Also, such a separate GUI could load data cubes and visualize them without having to run

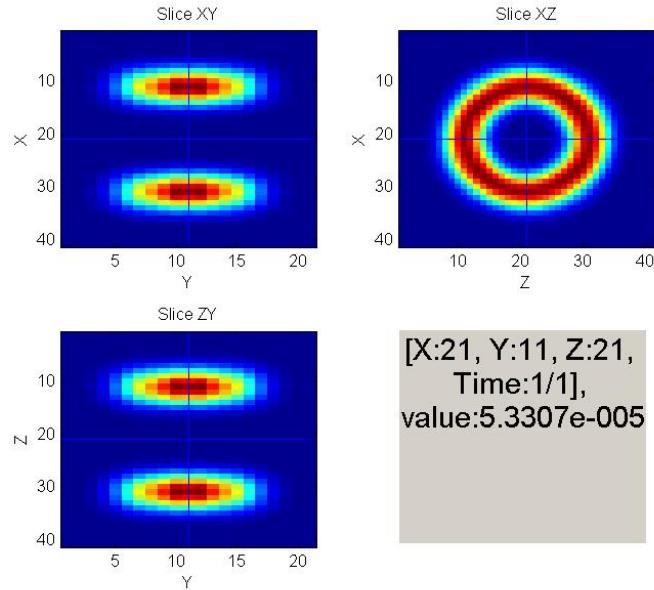


Figure 4: Example of slice plot during mid-run of a 3D vortex.

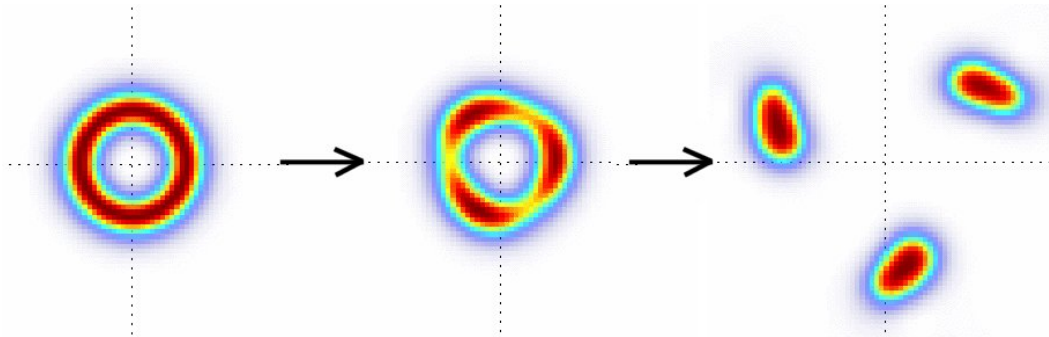


Figure 5: Example of azimuthal modulational instability of vortex in the CQNLS.

the integrator, and also add in isosurfaces, etc. There are some freely available MATLAB visualization GUI codes out there, some of which might be candidates for connecting to our program, but so far none are ideal.

Next, parallel implementation using MATLABs parallel toolbox would be very useful for large runs. Since we are using explicit finite differencing, the parallel implementation would, in theory, not be too difficult.

Finally, it would be a good idea to add adaptive gridding of some kind since for 3D problems, one wants to use grid points sparingly.

8 Conclusion

We have introduced a MATLAB GUI-based program for simulating the NLS using different finite differencing schemes and visualizations. Our program has been coded cleanly to allow users to add their own extensions tailored to their research with little or no difficulty.

We feel that our program could be very useful as a starting point, to be later enhanced by multiple users to become a standard option for integrating the NLS.

The program can be downloaded at: <http://www.sumseq.com/rmcNLS3D>

References

- [1] L. Debnath. *Nonlinear Partial Differential Equations for Scientists and Engineers*. Birkhauser Boston, New York, New York, 2 edition, 2005.
- [2] R. M. Caplan, R. Carretero-González, P.G. Kevrekidis and Q. E. Hoq. Azimuthal modulational instability of vortices in the nonlinear Schrodinger equation. *Optics Communications*, **282** (2009) 1399–1405.
- [3] Y. S. Kivshar and B. Luther-Davies. Dark optical solitons: Physics and applications. *Physics Reports* **298**, **298** (1998) 81 – 197.
- [4] J. C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. Society for Industrial and applied Mathematics, Philadelphia, PA, 2 edition, 2004.
- [5] G. H. Golub and J. M. Ortega. *Scientific Computing and Differential Equations An Introduction to Numerical Methods*. Academic Press, San Diego, California, 2 edition, 1992.
- [6] J.E. Castillo, J.M.Hymanb, M. Shashkov and S. Steinberg. Fourth- and sixth-order conservative finite difference approximations of the divergence and gradient. *Applied Numerical Mathematics*, **37** (2001) 171187.
- [7] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, New York, 2 edition, 2006.
- [8] R. Carretero-González, B. P. Anderson, P. G. Kevrekidis, D. J. Frantzeskakis and C. N. Weiler. Dynamics of vortex formation in merging bose-einstein condensate fragments. *Physical Review A*, **77** (2008) 033625.