THE UNIVERSITY OF ARIZONA

Simulating Vortex Manipulations with Complicated Pinning Site Arrangements

by

Logan L. Richardson

A thesis submitted in partial fulfillment for the
degree of Masters of Science

in the
The University of Arizona
Department of Optical Sciences
Under
Dr. Brian P. Anderson

July 17th 2013

Declaration of Authorship

I, LOGAN L RICHARDSON, declare that this thesis titled, 'Simulating Vortex Interactions with Complicated Pinning Site Arrangements' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

THE UNIVERSITY OF ARIZONA

ABSTRACT

Dr. Brian P. Anderson
Department of Optical Sciences

Masters of Science

by Logan L. Richardson

It is the work of this Master's thesis to show that we can transfer quantized vortices to pinning sites, where the pinning sites are arranged in an arbitrarily complicated manner. The study was undertaken using computational simulations of a BEC in MatLab. This is acheived by calculating an intial wave function and evolving it over time. External potentials were added to the BEC to generate pinning sites, and to create vortices.

Acknowledgements

I would like to thank Dr. Brian Anderson for his help and guidance with the subject. I really appreciated his patience and his tolerance with me asking how everything could be related to the gravitational quantum bouncer problem.

I would also like to thank my other commitee members for being able to review this thesis under little previous notice. Thank you Dr. Ewan Wright for your BPMPrimer paper, which helped me understand the core concept in the simulation.

I would like to also thank my parents for being supportive and helping me whenever they could. Thank you James for being there for me and my brothers when we needed you.

Contents

List of Figures

Chapter 1

INTRODUCTION

Vortices in Bose-Einstein Condensates (BEC) lead to turbulent behavior. One way to study how these vortices interact with each other is to manipulate and arrange these vortices within a BEC in a repeatable manner. The Ph.D dissertation of Edward Carlo Copon Samson [1] examines a method for generating and controlling vortices in a highly oblate BEC. In such a highly oblate dilute-gas BEC, fluid dynamics can be considered 2 dimensional. In Section 6.3 of his thesis, he experimentally studies method for creating vortices, and pinning them to specific locations through the use of external beams.

The goal of this thesis is to numerically extend Carlo's work to more complicated situations, and explore possible configurations for future experimental work. We achieve this by computationally simulating the dynamics of a Bose-Einstein condensate, introducing external potentials and generating vortices. We can turn on specific stationary external beams, and transfer created vortices to these stationary beams. The stationary external beams that the vortices are transferred to are known as a pinning sites. By transferring vortices to pinning sites we can create a prescribed array of vortices. By later turning off these pinning sites, we can watch how the system will evolve free of any external

interactions. The goal of this thesis is to evaluate the posibilites for the engeineering of complex vortex distirbutiosn in a BEC experiment.

We will discuss the methods used to generate these arrangements, as well as the problems that arose. We will assume the reader has a background in Bose-Einstein Condensation and is familiar with vortices in a BEC; See Refs. [2][3] for details on these subjects.

This thesis is broken down into four chapters, with subsections of each chapter devoted to specific topics. The four Chapters of the thesis are as follows:

Chapter 2 - Will discuss the overall methods for running the simulation, and a description of the code elements of the simulation.

Chapter 3 - Will provide the results and specific simulations we ran.

Chapter 4 - Will summarize the experimental results and provide insight into the future of the experiments.

Chapter 2

METHODS

2.1 Overview of Methods

In this chapter we will discuss the methods used to be able run simulations of a BEC. Due to the ease of repeatability, scaled vortex transfer experiments were simulated computationally. To accomplish this, we need an accurate way to generate and evolve BECs in a way that emulates the laws of physics. This section of the thesis is devoted to describing the methods used to simulate our BEC computationally, as well as how we were able to create vortices within our simulated BEC.

Our simulation routine can be broken down into two main sections:

1. The first section is concerned with creating an accurate initial wave function that represents what we would see under specified laboratory conditions. The programs HermGaus and BECShapeGen are used to accomplish this:

(a) **BECShapeGen -** Generates the shape required of our initial wave function based on trap parameters by solving the GrossPitaevskii equation using the Thomas-Fermi approximation [2]. To make the wave-function compatible with our evolution routine, we need to construct our wave function out of a superposition of Hermite-Gaussian basis states.

(b) **HermGaus -** Calculates an arbitrary number of Hermite-Guassian polynomials for BECShapeGen to use. Also establishes the scale of our initial wave function.

2. Once we are able to create an accurate initial wave function, we need to be able to evolve it in time. We achieve time evolution of our discrete wave function using the split-step method.

   (a) **Evolution and Simulation Code -** After an initial wave function is generated, we can evolve the function in time using the Split-Step Method [4]. Because we wish to simulate different experimental situations, this code is unique to the specific experimental simulation we are trying to demonstrate.

Figure 2.1 provides a general overview of the code used to simulate BEC vortex transfers.

Section 2.2 -A prominent challenge in accurately modeling the evolution of a Bose-Einstein Condensate is properly evolving an initial state. The split-step method is a good candidate for modeling wave function evolution. This method places a few restrictions on how we can set the scale of our model BEC, so we will start the discussion of methods with the split step method.

Section 2.3 - The wave function we are attempting to model must be discrete, due to the nature of computation. With the restrictions placed on our simulation due to the split

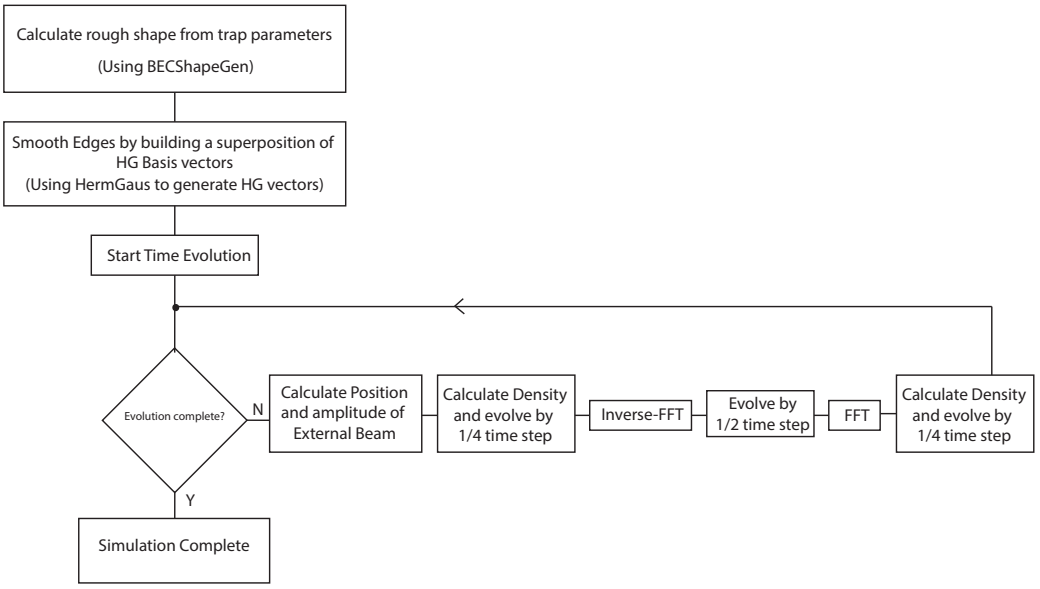

FIGURE 2.1: An outline of the BEC simulation

step method in mind, we can create a meaningful unitless scale to build our simulation on.

Section 2.4 - Once we have a scale to build on, we need to create our initial wave function. We solve the GPE using the Thomas-Fermi approximation to get the shape of our BEC using the function, BECShapeGen. This method however creates sharp edges which are incompatible with the split-step method of evolution. To circumvent this, we use the ideal shape to build up a super-position of Hermite-Gaussian basis states to smooth the edges.

Section 2.5 - Will discuss the generation of these Hermite-Gaussian polynomials used for smoothing the edges.

Section 2.6 - Once we have an intial wave function, and a method for evolution we can begin our simulation. This section will discuss the protocol for running simulations by incorporating the split step method.

Section 2.7 - After we have a stable 2D simulation of a BEC working, we can now perturb our system and introduce external beams. With these beams we can create vortices, pinning sites and alter other physical parameters of our BEC. This is the discussion of section 2.7.

2.2 The Split Step Method

Once we attain an initial wave function, we need to be able to computationally evolve our discrete wave function in time. Therefore, at the core of our simulation is our evolution routine. As previously mentioned, this method places on the restrictions on the nature of our discrete wave function.

The method we use to achieve time evolution is called the split-step method or the beam propagation method (BPM). Although there are other methods for wave function time evolution [5], it turns out that the split step method is generally straightforward to implement for our specific experiment.

The key to the split-step method is the separate evolution of the kinetic and potential parts of the GPE. As we will see below, it turns out that it is easier to evolve the kinetic energy portion in momentum space and the potential operator in position space. To see this, we first consider the ease of evolving the kinetic energy operator in momentum space. We start by analyzing the free particle (potential-less) Schrodinger equation in dimensionless units:

$$i\frac{\partial\psi(x,t)}{\partial t} = -\frac{1}{2}\frac{\partial^2\psi(x,t)}{\partial x^2} = \hat{T}\psi(x,t) \tag{2.1}$$

By using separation of variables, the solution is:

$$\Psi(x,t) = e^{-i\hat{T}t}\psi(x,0) \tag{2.2}$$

Which introduces the common time evolution propagator for a free particle. For a potential free situation, a time evolution of the wave function is solely under the influence of the kinetic energy operator. However, the problem lies in the fact that the kinetic energy operator has a second order partial derivative in it; which is computationally cumbersome to simulate.

The solution to this problem is illuminated when we consider the kinetic energy operator in both frequency and position space:

Position space $\rightarrow \hat{T}\psi(x,t) = -\frac{1}{2}\frac{\partial^2\psi}{\partial x^2}\psi(x,t)$

Momentum space $\rightarrow \hat{T}\psi(x,t) = \frac{iK^2}{2}\psi(x,t)$

In momentum space, the kinetic energy operator is free of derivatives.

To make the evolution operator more amicable, we introduce the fourier transform pair:

$$\psi(x,t) = \int_{-\infty}^{\infty} \phi(k,t)e^{-iKx}dK \tag{2.3a}$$

$$\phi(k,t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \psi(x,t)e^{iKx}dx \tag{2.3b}$$

Where $\phi(k,t)$ is the momentum space wave function. Using the information above, time evolution in momentum space can simply be written as:

$$\phi(k,t+\Delta t) = e^{-i\hat{T}\Delta t}\phi(k,t) = e^{\frac{-iK^2\Delta t}{2}}\phi(k,t) \tag{2.4}$$

Now if we want to understand this evolution in terms of position space, we can Fourier transform from momentum to back to position space. In this way we can we can write the evolution of the wave function in position space as:

$$\psi(x,t+\Delta t) = [FT^{-1}]e^{\frac{-iK^2\Delta t}{2}}\phi(k,t)[FT] \tag{2.5}$$

Where $[FT]$ represents a forward Fourier transform (eqn. 2.3a) and $[FT^{-1}]$ represents an inverse Fourier transform (eqn. 2.3b).

Now to understand how to evolve the potential operator, let us consider a time independant potential,in which the kinetic energy is zero. In that case the Schrodinger equation can be written as:

$$i\frac{\partial\psi(x,t)}{\partial t} = i\hat{V}(x)\psi(x,t) \tag{2.6}$$

Which admits the same type of solution as the kinetic equation 2.2.

In our specific simulation, the potential that we are trying to model has two terms, the harmonic trap, and the self-interaction term. To make things simple, we will exclude the interaction term for now [6] so the total potential can be written as $V(x) = 1/2x^2$. Plugging this into our propagator equation that we derived before, we directly get:

$$\psi(x, t + \Delta t) = e^{\frac{-ix^2\Delta t}{2}}\psi(x, t) \tag{2.7}$$

Which is relatively easy to evolve in position space.

It becomes clear that it is easy to evolve our potential term in position space, and our kinetic energy term in momentum space. The next task is to combine the two terms into a single expression.

The full Schrodinger equation can be written:

$$i\frac{\partial\psi(x, t)}{\partial t} = (\hat{V} + \hat{T})\psi(x, t) \tag{2.8}$$

Which solving as before gives:

$$\psi(x, t + \Delta t) = e^{\frac{-i(\hat{V}+\hat{T})\Delta t}{2}}\psi(x, t) \tag{2.9}$$

The problem is, we want to evolve the kinetic and potential terms separately. It would be convenient if we could separate potential and kinetic operators, as in:

$$'\psi(x, t + \Delta t) = e^{\frac{-i\hat{V}\Delta t}{2}}e^{\frac{-i\hat{T}\Delta t}{2}}\psi(x, t)' \tag{2.10a}$$

or

$$'\psi(x, t + \Delta t) = e^{\frac{-i\hat{T}\Delta t}{2}}e^{\frac{-i\hat{V}\Delta t}{2}}\psi(x, t)' \tag{2.10b}$$

so that we could evolve the potential in position space, and the kinetic term in momentum space. However this equation is invalid because $[\hat{T}, \hat{V}] \neq 0$. $\hat{T}$ and $\hat{V}$ may not commute, but the whole term including the timestep can approximately commute, in the limit of a small enough time step:

$$\lim_{t \to 0}, [\delta t\hat{T}, \delta t\hat{V}] \to 0 \tag{2.11}$$

Because for a small enough time step, $\delta\hat{V}$ and $\delta\hat{T}$ go to zero, which allow for commutation to occur. Since commutation doesn't dictate whether or not our equation should be 2.10a or 2.10b. To make the equation symmetric then, we can write our evolution operator as:

$$e^{\frac{-i(\hat{V}+\hat{T})\Delta t}{2}} = e^{\frac{-i\hat{T}\Delta t}{2}} e^{-i\hat{V}\Delta t} e^{\frac{-i\hat{T}\Delta t}{2}} \tag{2.12a}$$

or

$$e^{\frac{-i(\hat{T}+\hat{V})\Delta t}{2}} = e^{\frac{-i\hat{V}\Delta t}{2}} e^{-i\hat{T}\Delta t} e^{\frac{-i\hat{V}\Delta t}{2}} \tag{2.12b}$$

With our knowledge of how to evolve the kinetic and potential terms, we now have a clear recipe on how to evolve our wave function through one time step. Using our

expression for kinetic energy evolution (eqn. 2.5) we get:

$$e^{\frac{-i(\hat{V}+\hat{T})\Delta t}{2}} = [FT^{-1}]e^{\frac{-i\hat{T}\Delta t}{2}}[FT]e^{-i\hat{V}\Delta t}[FT^{-1}]e^{\frac{-i\hat{T}\Delta t}{2}}[FT] \tag{2.13a}$$

or

$$e^{\frac{-i(\hat{T}+\hat{V})\Delta t}{2}} = e^{\frac{-i\hat{V}\Delta t}{2}}[FT^{-1}]e^{-i\hat{T}\Delta t}[FT]e^{\frac{-i\hat{V}\Delta t}{2}} \tag{2.13b}$$

As we can clearly see, both equations 2.13a and 2.13b are equivalent, but eqn. 2.13a, requires two more Fourier transforms than eqn. 2.13b. From a computational point of view, equation 2.13b is more lucrative since Fourier transforms are a computationally costly operation.

A criterion of this method is to be able to perform Fourier transforms. Computationally, these are carried out as fast Fourier transforms (FFT). A requirement of the FFT is that the discrete scale we choose to represent our wave function must have a resolution of the form 2^n where n is any non-zero integer.

2.3 Scale and Units

2.3.1 Time step

Now that we understand the requirements of our evolution mechanism, we can discuss the scale and parameters the determination of what value of the time step t is small enough. In [**?**] it is worked out that a time step that is small enough will have the form:

$$\Delta t < \frac{2x_{max}^2}{N^2\pi^2} \tag{2.14}$$

Where x_{max} is the maximum value of the x scale and N is the number of points on the scale. These values of these parameters are determined in section 2.5 of this chapter.

2.3.2 Position Scale.

A feature of computational simulation is that we cannot continuously model our BEC, but rather we need to model the wave function discretely. To accurately sample our BEC we need to designate a proper length scale. Another condition on this scale is that we require it to be unitless. To get a unitless scale we define our position in terms of harmonic oscillator lengths. Congruent with the typical definition[7], we divide by natural harmonic oscillator length to get a unitless scale. The harmonic oscillator length is defined:

$$\sigma = \sqrt{\frac{\hbar}{m\omega}} \tag{2.15}$$

m is the mass of the particle and ω in our experiment corresponds to the trap frequency. We will discuss the specifics of the experimental parameters in the next section, but for our experiment we use 87Rb with, $m = 1.442 \times 10^{-25} kg$ and trap frequency equal to $\frac{\omega}{2\pi} = 8Hz$. Plugging in these values we a harmonic oscillator length of $\sigma = 1.1^{-6}m$

We can divide our scale by the parameter σ, to get a unitless position scale.

2.4 BEC Intial Wavefunction Shape Generation

This section is devoted to creating the intial wavefunction we evolve. Excerpts from the program BECShapeGen will be referenced in code boxes. The full program is available in A.1.

As noted before, our BEC wave function can be modeled using the Gross-Pitaevskii equation. To solve the GPE for our initial wave function, we will use the Thomas-Fermi approximation, which neglects the kinetic energy term[2]:

$$[V(r) + g|\psi(r)|^2]\psi(r) = \mu\psi(r) \tag{2.16}$$

Where g is the interaction term, and μ is the chemical potential.

Solving for ψ we get:

$$\psi(x, y = 0) = \begin{cases} A\sqrt{(1 - \frac{x^2}{x_0})} & \text{for } |x| \leq x_0, \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

Where A is the normalization constant, and X_0 where the chemical potential and harmonic potential intersect. Within this equation only two things can affect the shape our initial wavefunction; the potential due to the harmonic trap, and the interaction term g.

By altering the frequency of the trap, we can affect changes to the shape of our BEC. We define the trap frequencies in terms of those that we see in typical experimental setups. The X and Y trap frequency are the same.

```
%%%%%%%%%%%%%%%% PHYSICAL PARAMETERS %%%%%%%%%%%%%%%

%Here we set Sigmax = Sigmay. If we want to change this (make an oblate
%BEC) here would be spot to change things.

%Trap Frequencies
sigmax = 3.74E-6;   %8 hz
sigmay = sigmax;
sigmaz = 3.5E-6;   %effective trap thickness due to interactions
```

Beyond the harmonic potential, another factor that determines the wavefunction of our BEC is the interaction term. The interaction term is dictated by:

$$g = N\sqrt{8\pi}\frac{a}{\omega_z} \tag{2.18}$$

Where, N is the number of particles, a is the scattering length, and ω_z is the z-trap frequency.

We define these values, and for the rest of experiment they remained unchanged. N and σ_z were determined in comparison to experimental data. The scattering length value, a, for $87Rb$ is a= 5.5nm

```
scatlen = 5.5E-9;
NumPar = 1e6;

g2D_dimensionless = sqrt(8*pi)*scatlen/sigmaz;
g2D = NumPar * g2D_dimensionless;
```

We define our scale in terms of the previous chapter. The chemical potential is defined in the program as $E = \sqrt{g2D}$. We can solve for the classical turning points by subtracting our potential from the chemical potential. The points where they overlap correspond to the classical turning points. With all the pieces in place, we can solve for ψ by using equation 2.17, and then normalizing.

```
%%%%%%%%%%%%%%%% BEC ROUGH SHAPE DEFINITION %%%%%%%%%%%%%%%%

E = sqrt(g2D);   %chemical potential in dimensionless units
CPT =  sqrt(2*E);
```

```
BECXY = sqrt(1-(Sigx/CPT).^2-(Sigy/CPT).^2); %Defines the shape of the BEC
BECXY = BECXY .* (BECXY > 0);

A = sqrt(sum(sum(abs(BECXY).^2)))*dx;
BECXY = BECXY/A;
```

When we use this method to create our initial wave function, we get sharpe edges. These sharp edges, can lead to problems. These problems will be discussed in the next section in detail.

To rectify this we can smooth out the edges by building the rough shape out of a series of Hermite-Guassian Polynomials. By building up a series of HG polynomials, the edges will be smoothed out and appropriate for evolution[8].

To do this, we run the code HermGaus (full explanation of code in next section). We specify the number of basis states we want to use, and the number of sample points we choose to use. This program returns an array where every row is a basis state HG polynomial.

We then take the first HG polynomial, and create a 2D array of that basis function. We then take that 2D function, perform a 2 dimensional integral over the dot product of the 2D basis function with the 2D Rough BEC shape we are trying to build. These values are stored in the variable Coef2. We then add up our coefficients multiplied by our basis state to build up our BEC function.

```
%%%%%%%%%% COEFFICENT CALCULATION AND SMOOTH DEFINION %%%%%%%%%%%%%%%%%

BECINIT = 0;
for(i = 1:Ender)
    for(j = 1:Ender-i)
    HERMGAUSNOW = transpose(H(i,:))*(H(j,:));
    Coef2 = sum(sum(HERMGAUSNOW .* BECXY*dx*dy));
    %Coef2 = sum(transpose(Coef1))
    BECINIT = BECINIT + Coef2 * HERMGAUSNOW;

    end

end


A = sqrt(sum(sum(abs(BECINIT).^2)))*dx; %Norm
```

```
BECINIT = BECINIT/A;
```

After we are done, we normalize and now we have generated our 2D initial wavefunction.

2.5 Hermite-Gaussian Calculator

This section is devoted to the section of code called HermGaus. This function calculates an arbitrary number of Hermite-Gaussian (HG) basis states for a specific scale. The full code for this program is available in Appendix A.2.

As discussed in the previous chapter, to get the basic shape of a BEC, we subtract the potential from the chemical potential. This provides the proper shape, but creates a couple of issues. If our system contains sharp edges, it can create problems with proper evolution. The problem is that these sharp edges correspond to large energies in the system, which will cause improper evolution. To smooth out these edges we build the shape of our BEC out of a linear superposition of Hermite-Gaussian (HG) polynomials.

The program HermGaus inputs an order number and number of sample points (denoted: mx) to create a matrix of polynomials up to the order number, with spatial resolution equal to the number of sample points. The program starts by defining σ, which was calculated in section 2.3.

A scale is defined from -20 to 20 units of rest length, with a number of points mx +1. The values -20 to 20 are somewhat arbitrary. The size of our BEC is completely determined by factors such as particle number and trap energy. Based on the conditions we chose to simulate, -20 to 20 gives us enough space to work in. If the scale is too small, the program will clip the edges of the BEC giving rise to spurious effects and not effectively modeling the BEC.

We define the scale using the linspace command. This command creates a number of evenly spaced points between the two scale values (-20 to 20). The number of points in this interval must be of the form 2^n, so that it is compatible with the fast Fourier transform (FFT), as discussed in the split step section. Another requirement of the FFT is that we clip the last value to avoid aliasing issues. To account for this, we demand that mx is of the form 2^n, we set the number of points in the linspace command equal to mx $/ + 1$ and then clip the last value. Now we have a clipped scale that is compatible with a FFT.

It turns out the best way to generate HG polynomials is by using a recursive relationship. By starting with a seed equation, and using the recursive relationship we can create any order of HG polynomial we require.

There are a couple different definitions of HG recursion relationship. We will use the 'physicists' recursive relationship for Hermite polynomials, and multiply on the Gaussian section. The Physicist's Hermite recursion relationship is defined by[9]:

$$H_{n+1} = xH_n(x) - 2nH_{n-1}(x) \tag{2.19}$$

where n is the order, and H_{n+1} is the next polynomial in the series. To make this a Hermite-Gaussian, we multiply our Hermite polynomials by a Gaussian. Since the recursion relation utilizes uses past Hermite-Gaussian to generate the next order, we define the zeroth and first order HG polynomials for the recursion relation to use.

The program starts with the zeroth order HG polynomial is a constant multiplied by a Gaussian with a harmonic oscillator length sigma. The second is the same Gaussian multiplied by our linear scale x. These polynomials are stored in a matrix, where the

row number corresponds to 1 more than the order number and each element in that row corresponds to a value of that function at that position.

For example, for the zeroth order polynomial, its row number is 1, and its first element corresponds to the value of the Gaussian evaluated at -20; the last element of this row corresponds to the value of the Gaussian at x = 20;

The program then goes into using the recursion relation in a loop to calculate the next generation of HG polynomials. Each new HG is put into a new row. The loop terminates when it reaches the order of polynomial desired.

2.6 Simulation and Evolution

Now that we have a method for calculating our initial wave function, we can apply the split-step method to evolve our wavefunction in time

We first start our program which runs the TwoDBECGenCorrect code. This will return our initial wave function that we hope to evolve. It also returns the scale that we define in the previous program, as well as the g parameter.

One of the more crucial parameters in this code is our time step. As we discussed with the split step method section, if the time step parameter is too large, it will not properly simulate time evolution, if it is too small, the simulation will take a lot longer to run. For some of the complicated vortex arrangements that we want to simulate, this additional time can be compounded and make a simulation take a very long time to run.

The movemax terms determine how long each stage of time evolution occurs. This becomes important because certain actions, such as turning on beams, or moving beams can depend on how quickly these events happen. As we will discuss later, moving beams

too quickly can result in free vortices to be released, too slowly and vortices will not be created. The rates at which these things happen is tied into these movemax values.

Due to the nature of Matlab, functions that need to be Fourier transformed need to be fftshifted, before and after a Fourier transform. Because fftshift terms can be computationally costly we, arranged the scale in a way so that it doesnt need to be FFT-shifted.

Below that we setup a scale in frequency space in a similar manner, and we Meshgrid this to create a 2D frequency space.

```
% X scaling.
dx = xmax/N;
nmid = floor (N/2);
v0 = [0:N-1];
x = (v0 * dx) - (xmax/2);
y = x;
[X,Y] = meshgrid(x,y);

%k Scaling.
kmax = (2 * pi)/(dx);
dk = kmax/N;
p = find(v0 > nmid);
vp = v0;
vp(p) = (N - v0(p));
eta = vp * dk;
nu = eta;

[Eta,Nu] = meshgrid(eta,nu);
```

We then set our initial wave function equal the output of the BECShapeGen.

Within our for loops, we need a method to be able to keep track of the iteration number. To accomplish that, we have a variable called ticker at the beginning of every for-loop, we reset this to zero. During the course of one iteration, we add 1 to this value. By dividing the ticker value by the movemax value for that specific for loop we can get a percentage of how far along is the loop along in the program. We can use this fraction to move beams or turn beams on.

The first thing we need to calculate is the density term. Since in the GPE a term depends on the density of the wave function at each point, we can calculate it by absolute

magnitude squared of the wave function. By multiplying this density by the g term we calculated above, we now have the self-interaction term within the GPE.

```
density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;
```

We also include a dampening term. This term will help us settle the BEC into its true ground state, which will be discussed in depth in the next section.

In the case where the interaction term is the only potential (not accounting for external beams yet), once we calculate the interaction term, we can evolve the system by a factor of t/4.

```
Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
```

From here we can transform into momentum space and evolve by a half time step:

```
%Transform to momentum space


Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
```

Then we transform back into position space by the use of a forward FFT. We then re-calculate the density for the interaction term, evolve for a fourth of a time step.

```
%Transform back to free space

Psi2 = fft2(Phi0);


density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
```

PsiOut, is a wave function that has now been evolved by one complete timestep. We then re-normalize and image the density of our wavefunction:

```
A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;
```

```
P2 = abs(PsiOut.*conj(PsiOut));
p02 = abs(Psi0.*conj(Psi0));
imagesc(x,y,P2);
%plot(x,P2)
drawnow
```

This evolution repeats in the for loop until it reaches the movemax value. We now have a BEC under a static potential, which evolves in time from an initial wave function. In the next section, we will introduce an external potential to alter this wave function so that we may simulate vortices.

2.7 BEC Manipulation

In the previous section, we established a method for evolving our initial wave function. To be able to generate vortices, we need to apply external potentials, alter the dampening and move these external potentials through our BEC. This section will discuss how we achieve this.

In order to run a specific simulation, we will run an evolution module that performs some manipulation on our BEC over time. Different modules correspond to different laboratory events.

There are three primary ways we can manipulate our BEC:

1. Altering the damping of our BEC.
2. Introducing or removing an external potential.
3. Moving that external potential within our BEC and creating vortices.

From these three types of manipulation, we can build up and create our scaled up potential beam setup.

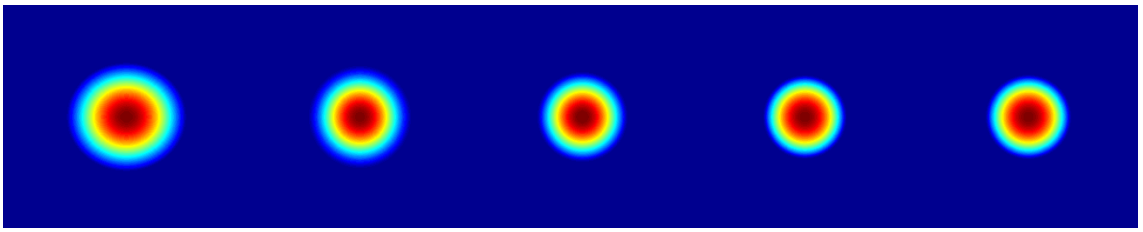

FIGURE 2.2: The figures above show sseveral frames of a BEC lined up side by side. The colors correspond to the density of each BEC at any given loctation. The BECs radius decreases and increases in an osscilatory manner until it comes to rest in its true ground state.

2.7.1 Damping and Ground State Relaxation

Although we have calculated an accurate initial wavefunction, it is not the exact ground state wavefunction. Due to this small difference between what we calculated, and the actual ground state, our BEC will oscillate between the two states. To stop this oscillation, we can introduce damping terms.

With a damping term, our BEC will settle into its true ground state. For the BEC to settle, we just allow it to evolve in time with a damping term added to the evolution propagator. Over time, the oscillations will stop, and the BEC will be in its true ground state. Figure 2.2 shows the system settling into its ground state.

This damping term will affect our BEC manipulation techniques, so we turn down the damping. To do this, we have a module of the simulation that lowers the value of the damping term over time. The beginning of every simulation will contain these two modules.

2.7.2 External Beams

In the course of our simulations, we will need external beams to simulate pinning sites, and to create vortices. A simulated beam is a potential just like any other. This allows

us to add the potential of the beam directly to our potential profile defined above.

We need to firstly define the profile of the beam. For the purposes of this thesis, we used Gaussian beams. The amplitude of the beams were defined in terms of the energies calculated above.

The general form of the beam is:

$$V_{Laser} = V_{Max} e^{\frac{-2(X-x_0)^2+(Y-y_0)^2)}{\sigma^2}} \tag{2.20}$$

Vmax corresponds to the amplitude, which is written in terms of μ. x_0, y_0 refer to the positions of the center of the beam, which may be time dependant, σ is is the beam width.

We can add the beam potential directly into eqn 2.13b for our evolution. We do however need to slowly introduce the beam potential so that the BEC has time to adjust to the external potential. To do this, we slowly ramp on the amplitude of the external beam.

Each for loop's ending parameter is known as movemax in the code. By introducing a term that corresponds to how many iterations have taken place (which we call ticker), and dividing by movemax, we can get a percentage of how far through the module any given iteration is. By multiplying our beam term in eqn. 2.20 by $\frac{ticker}{movemax}$, as the BEC evolves in time, the beam will be ramped on gradually.

An example of a couple of beams being gradually turned on for pinning sites can be found in an exerpt of appendix A.4 in the simulation discussed in section 3.2:

```
%Generation of third and fourth Laser

ticker = 0;
for(k =1:movemax5);

    x3t = 2;
```

```
    y3t = 2;
    x4t = 2;
    y4t = -2;
    x5t = 0;
    y5t = 4;
    x6t = 0;
    y6t = -4;

Vmax = 1.2*E;
dtild = .8;

Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtild^2))*ticker/movemax5;
Vlaser4 = Vmax*exp(-2*((X-x4t).^2+(Y-y4t).^2)/(dtild^2))*ticker/movemax5;
Vlaser5 = Vmax*exp(-2*((X-x5t).^2+(Y-y5t).^2)/(dtild^2))*ticker/movemax5;
Vlaser6 = Vmax*exp(-2*((X-x6t).^2+(Y-y6t).^2)/(dtild^2))*ticker/movemax5;

Vlasers = Vlaser3+Vlaser4+Vlaser5+Vlaser6;
ticker = ticker + 1;

%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;


i2 = 1i-damp2;
Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);

%Transform to momentum space

%Phi0 = fftshift(fft(fftshift(Psi1)));
Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);

%Transform back to free space

%Psi2 = fftshift(ifft(fftshift(Phi0)));

Psi2 = fft2(Phi0);


density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);



A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;



P2 = abs(PsiOut.*conj(PsiOut));
p02 = abs(Psi0.*conj(Psi0));

%Graphing Area

subplot(1,2,1)
xlabel('x Position [microns]');
ylabel('y Position [microns]');
title('Density')
imagesc(x,y,P2);
axis image
subplot(1,2,2)
anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
imagesc(x,y,anglef);
xlabel('x Position [microns]');
ylabel('y Position [microns]');
title('Phase');
axis image
```

```
drawnow


end
```

2.7.3 Moving Beams and Vortex Creation

Moving the beams becomes critical for vortex creation. The dynamics of vortex creation can be found in [1].

To move the beams we can alter the values of x_0 and y_0 in eqn. 2.20. By doing this gradually, we can simulate movement of the beams in our BEC. To do this, we change the values of x_0 and y_0 by a small amount each loop iteration by using $\frac{ticker}{movemax}$. We can set an intial beam location value, and a final one, with and move the beam a fractional amount each iteration. When changing the value of Movemax for that specific loop, this is in turn changing the rate at which the beam moves through the BEC. The larger the value of movemax, the slower the beam moves.

To create vortices we need to move beams through the BEC. Vortices must be created in pairs. To create a pair of vortices, we overlap two beams, and then move the beams away from their location, at different angles. As the beams push fluid around them, they create vortices.

If the vortex generation beam separation is too slow, it will not generate vortices. Moving them too quickly can generate free vortices. Free vortices are vortices that aren't pinned to a beam, and move through the BEC without any external influence. These free vortices can mess with the simulation, so limiting the number of them is ideal. Figure 2.3 shows the creation of free vortices.

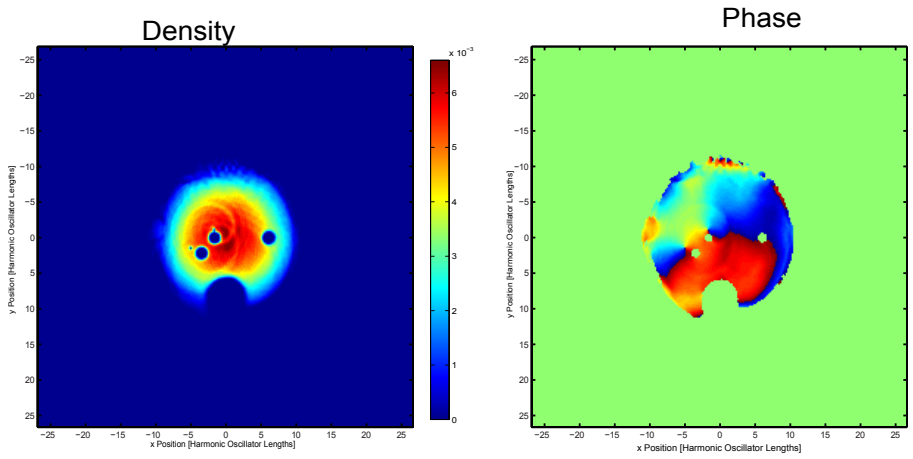

FIGURE 2.3: The left plot above shows the density profile of a BEC, where the colors represent the relative density at each given location. In the density plot above we can see two holes in our BEC that are caused by external beams. The beams in particular have been moved too quickly and have generated free vortices. These vortices can be seen in the phase diagram above.

Chapter 3

RESULTS

3.1 Introduction

Within this chapter, we will explore some of the simulations and more complicated arrangements using the results from the previous chapter. Each simulation's code can be found in the appendix.

Overview of this chapter:

Section 3.2 - The transfer of four vortices to four separate pinning sites.

Section 3.3 - The transfer of one vortex to a pinning site, while one vortex is taken out of the BEC

Section 3.4 - A central pinning site with multiple vortices

Section 3.5 - Five vortices transferred to two sites

Section 3.6 - A large number of singularly polarized vortices

Section 3.7 - Circular vortex distribution along with an annular beam distribution.

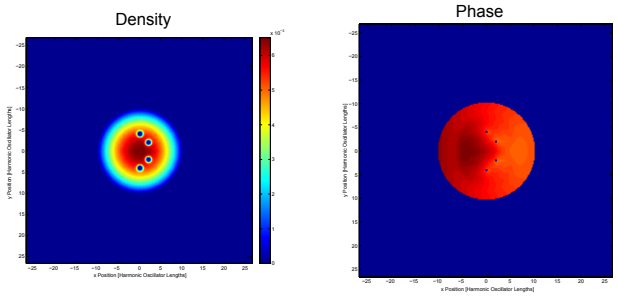

FIGURE 3.1: This figure shows the pinning site arrangement for our first simulation. The left plot is the density profile of our BEC, and the right plot is the phase plot. The pinning sites are the spots of low density, where an external beam has cleared the fluid from a specific location.

3.2 Four Vortices transferred to Four Pinning Sites

This first simulation, although simple, provides insight into how the more complicated experiments work. The simulation places four vortex cores in four different pinning sites. The top two vortices will be of one charge, and the bottom two pinning sites will contain vortices of the opposite vortex charge. The code for this program can be found in appendix A.4.

We setup initial conditions as we described in Chapter 2. We setup the scale and create the initial wave function using BECShapeGen and HermGaus. After intializing the parameters of the program, the first thing we do is simply evolve the wave function to allow it to settle into its ground state. The damping parameter allows the difference between the ground state and our initial wave function to settle. After our wave function has settled into its ground state, the next module turns down our damping.

Once our BEC is prepared, we can now start the actual simulation. The first thing we do is turn on the pinning sites. We have four pinning sites, which we turn on simultaneously. Figure 3.1 demonstrates the location of our pinning sites.

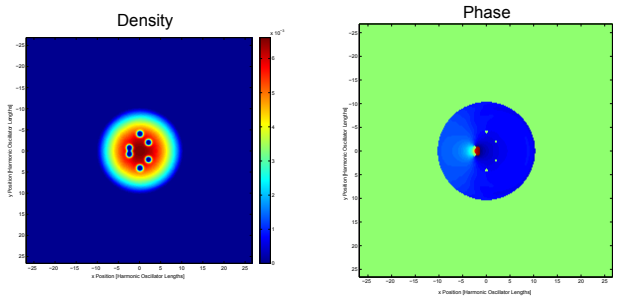

FIGURE 3.2: First Vortex Pair Creation.

Once we have the pinning beams on, we can start generating our vortices two at a time. We turn on two beams that overlap on a beam generation site. The beams are at 1.2μ each, and they are overlapped. All information about beam parameters can be found in the appendix.

The next part of the module is concerned with creating the first vortex pair, and then moving the two beams to the pinning sites. The key to this is ensuring that when the two beams are separated that we generate two vortices of opposite charge. The key to this is the rate of separation and the angle. As discussed in section 2.7.2, if the beams are moved too quickly, we will create free vortices. Figure 3.2 demonstrates the creation of our first set of vortices pinned to the generation beams. They show up clearly in the phase part of the plot, which is the graph on the right.

As the beams come in contact with the pinning sites, the vortices from the moving beams are transferred to the pinning sites. The first vortex beam transfer is demonstrated in figure 3.3.

Once the vortices have been successfully transferred, we can ramp down the carrying beams and ramp up two new vortex generation beams at the same location that the first pair was created. We pull them apart in a similar manner as the first pair. The second vortex pair creation is shown in figure 3.4.

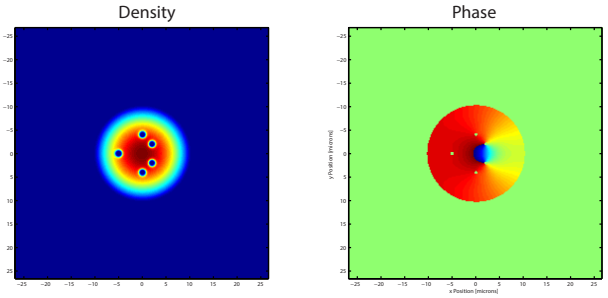

FIGURE 3.3: First Vortex Pair Transfer.

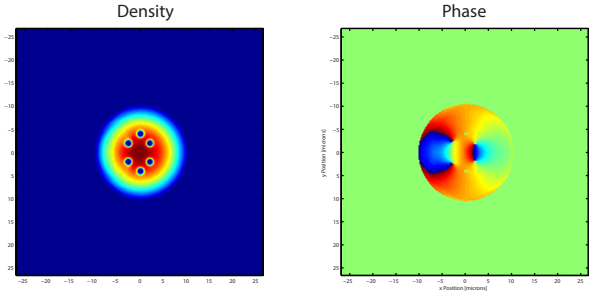

FIGURE 3.4: Second Vortex Pair Creation.

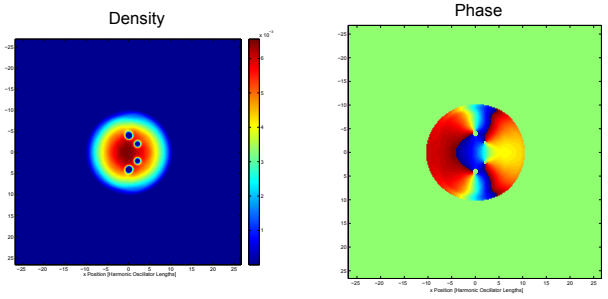

FIGURE 3.5: Final Vortex Beam Transfer.

Finally we transfer the second set of beams in the same way as the first pair. Figure 3.5 shows all four vortices transferred to each pinning site.

After all four vortices have been pinned to the desired pinning locations, we have completed our simulation.

From this simulation we learn the basics of creating and manipulating vortices within a BEC. We can see how vortices are transferred to each pinning site, and the methods for not creating free vortices.

3.3 Pulling a single vortex out of a BEC, placing one in a Pinning site

This simulation concerns getting rid of vortices that we do not want in our BEC. Removing vortices from a BEC can cause a whole host of issues, this simulation will provide some insight into some of these problems, and how we can address them. The code for this simulation is available in appendix A.5.

For the rest of the thesis, vortices we wish to remove will be referred to as dismissed vortices (DV), whereas vortices bound for a pinning site will be referred to as pinning vortices.

In this simulation we transfer a vortex to a single pinning site, while removing the other vortex from the BEC.

As mentioned in section 2.7.2, for any given vortex creation there is a range of timestep values that constitute a happy medium. These timestep values constitute a beam separation rate that is not too fast (so it does not generate free vortices) and not too slow (not creating vortices). Depending on the geometry of the beam separation, typically this happy-medium has a range of 200 timesteps.

When pulling a vortex out of a BEC, the vortex creation process can be tricky. When pulling one vortex out, and keeping one in, the vortex creation becomes a lot more sensitive to the rate at which the beams are moved. The reason for this is that, the difference in path lengths that the two beams need to travel can be vastly different. So for any given beam movement rate, one beam might be moved too slowly to generate vortices, while the other might be moved too fast and create free vortices. In other words, instead of being able to alter the timestep values to any value within a 200 value range, we have to find a value where both happy mediums overlap. The greater the difference between these path lengths, the smaller the range of acceptable timesteps is. In more complicated codes, the path separation will become a much larger issue, but for now we can discuss the four ways to address this problem.

1. Finding the timestep that satisfies both beams - This is the most straightforward way to change things, but can be hard to find. Essentially, by changing the rate at which the beams move, so that the rate isn't too quick for one beam or too slow for another.

2. Changing where the dismissed vortex leaves the beam, making the difference in pathlengths smaller. However, using this method, beam geometry can be affected making it more difficult to create vortices.

3. Creating two separate path motions. This one is the easiest to implement and typically the safest to work with. Essentially it works by moving the dismissed vortex to a safe location within the BEC as the pinning vortex is transferred to its pinning site. Once the beam is transferred, the next set of code takes the dismissed vortex out of the BEC.

4. Changing the rate at which the beams are moved to a non-linear motion. The most tricky to implement, but instead of moving the beams at a constant rate, by moving

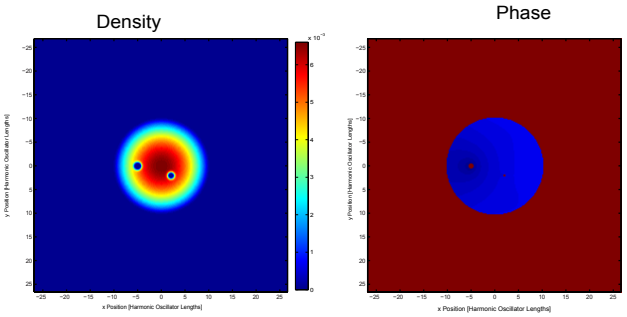

FIGURE 3.6: Pinning Site Arrangement and Vortex Creation Beams. In units of harmonic oscillator length, vortex creation beams are located at (-5,0), the pinning site is located at (2,2)

them at an exponential or logarithmic rate can allow a beam to be separated slowly, but then taken out more quickly.

In most instances a combination of the first three methods are used to generate vortices. In the simulation discussed in section 3.6, a non-linear beam separation method was used, due to the nature of how the beams interacted with the edge reservoir. In this specific simulation we will focus on the use of the 3rd method by creating two separate paths.

As before, the program allows the BEC to settle into the ground state. Then the BECs damping is turned down. After that the singular pinning site is turned on, as well as the two overlapping vortex creation beams. Figure 3.6 shows the pinning site and two vortex generation beams being turned on.

As before, we separate the two beams to create vortices. We will be moving the dismissed beam to within the BEC in this segment of code, so that we can easily generate vortices. Attempts to remove the vortex directly were difficult to achieve, and so we will remove the dismissed vortex in two steps. Figure 3.7 shows the separation of the two beams, successfully generating two vortices.

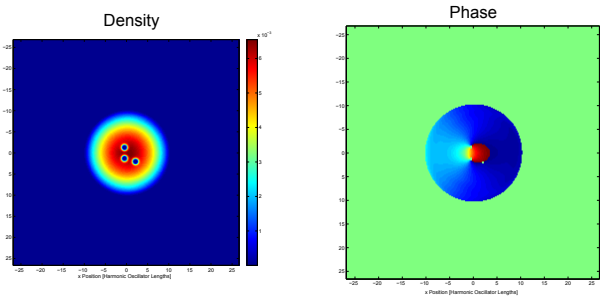

FIGURE 3.7: Separation of initial vortex pair.

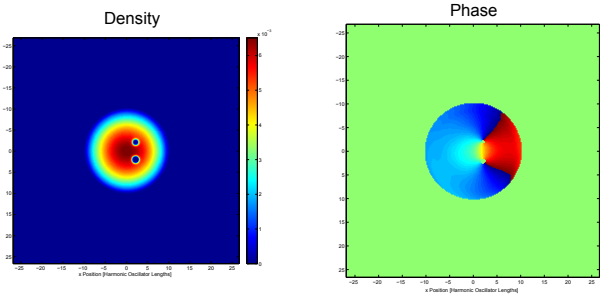

FIGURE 3.8: The vortex is transferred to the pinning site, while the dismissed (upper) vortex changes trajectory to leaving the BEC.

One vortex is transferred to the pinning site. The dismissed vortex remains unpinned over a safe location. Figure 3.8 shows the exchange. The next segment of code moves the dismissed vortex out of the BEC while leaving the pinned vortex in its pinning site.

As the dismissed vortex beam is brought to the edge of the BEC it begins to create disturbances near the edge of the BEC. This can be seen in figure 3.9. As the dismissed beam is brought out of the BEC, the vortex is transferred to the edge of the BEC. This edge vortex will circle around the edge of the BEC for the remainder of the experiment. As long as there are not any pinning sites near the edge, this edge vortex will not disturb anything. Figure **??** shows the motion of the edge vortex along the BEC.

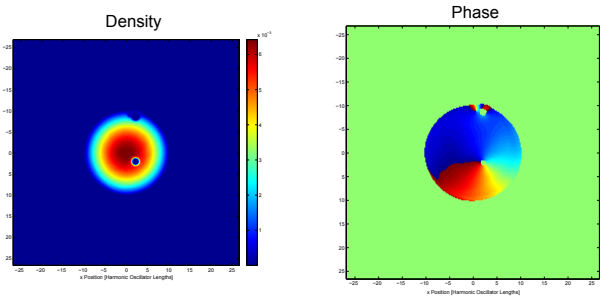

FIGURE 3.9: The vortex is brought to the edge of the BEC, creating a disturbance that travels around the edge of the BEC.

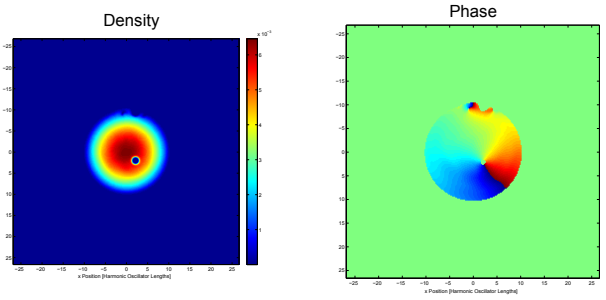

FIGURE 3.10: The dismissed vortex beam transfers its vortex to the edge of the BEC

3.4 Ten Vortices Transferred to Two Pinning Sites.

In this simulation, we will show that we can transfer multiple vortices to the same pinning site. We will create 2 pinning sites, and transfer five vortices of the same charge to each pinning location. The code for this simulation can be found in appendix A.6.

It is known that the number of vortices a pinning site can hold is related to the geometry of the pinning site [10] . The geometry of our current pinning sites (Depth = 1.2 μ, width = .8 harmonic oscillator lengths) is only capable of holding two vortices. If we try and add more than two vortices to our pinning sites, a free vortex will leak out. A demonstration of free vortex leakage is shown in figure 3.11.

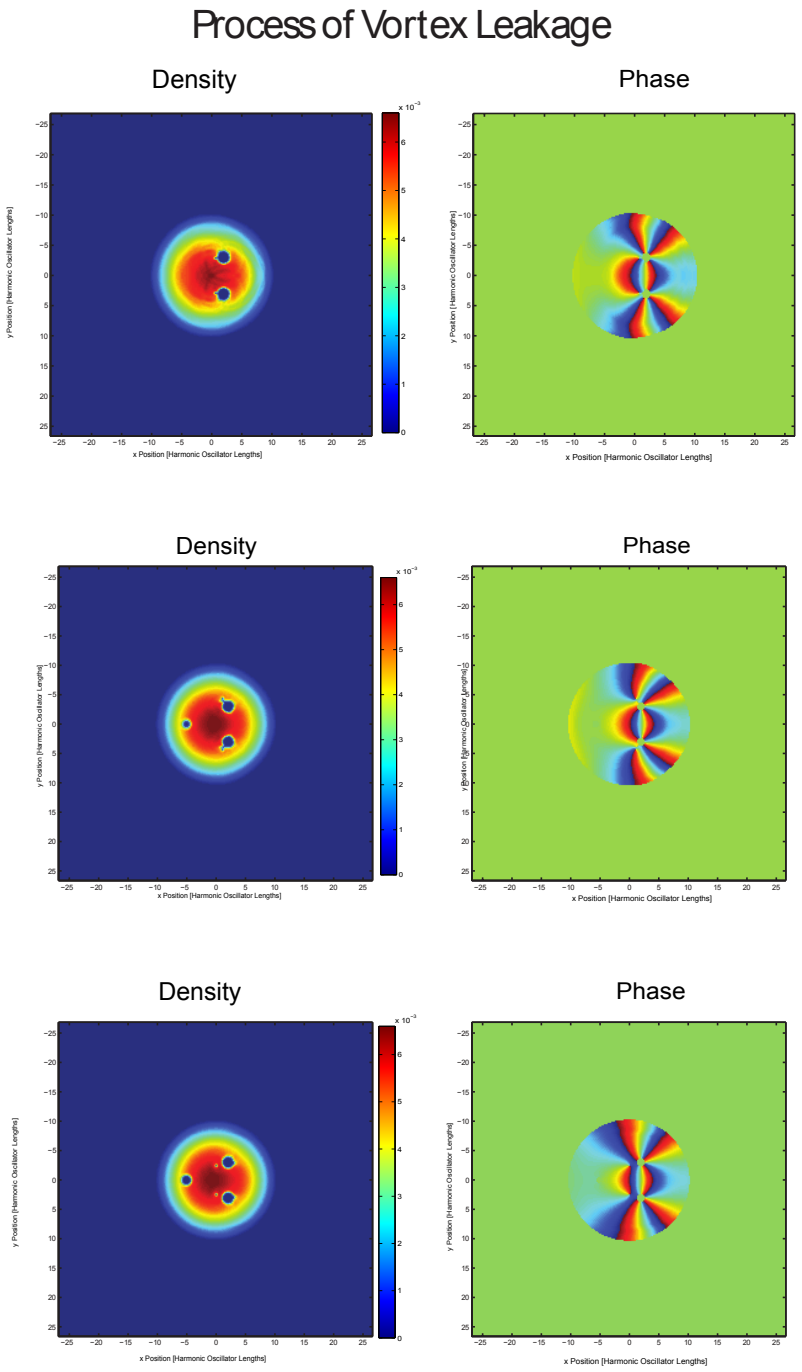

FIGURE 3.11: The process of vortex leakage, when a pinning site is full. The images progress from left to right, then top to bottom. The phase diagram is to the left of each density figure. When a pinning site is full, the addition of another vortex causes the site to leak a vortex.

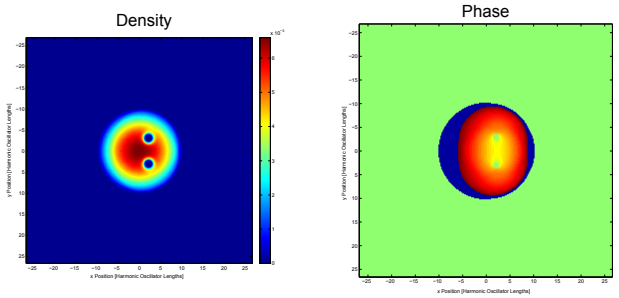

FIGURE 3.12: New Pinning site sizes.

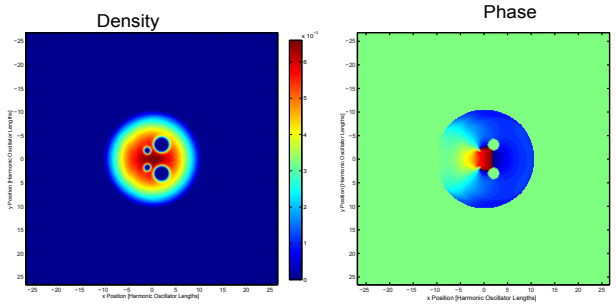

FIGURE 3.13: First vortex pair heading towards pinning sites.

To accommodate more than two vortices, we need to increase the amplitude of the external beam, or its width. In this simulation, we increase the width to 1.4 harmonic oscillator lengths, and the amplitude to 3.6 μ.

After the usual ground state settling and damping reduction, we turn our pinning sites with their new parameters. This is shown in figure 3.12.

These new pinning sites will be able to hold at least five vortices. After the pinning sites were turned on, we generate vortices in the usual manner. Figure 3.13 shows the first vortex pair being brought towards our two pinning sites.

Since the ramping on of the vortex generation beam, the creation and movement of the vortices is the same for each transfer, this simulation utilizes a for-loop for each cycle.

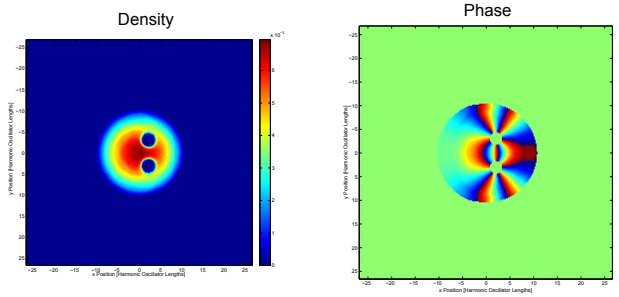

FIGURE 3.14: Fourth vortex pair being transferred.

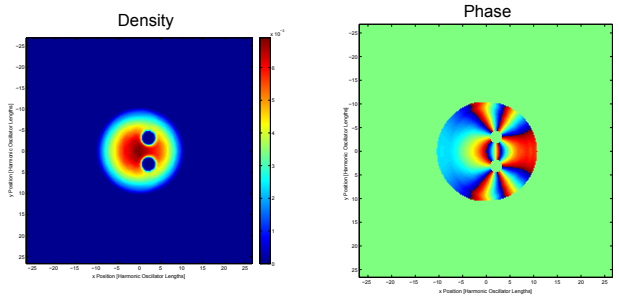

FIGURE 3.15: Five vortices in each pinning site.

The program keeps transferring vortices to the pinning sites. Figure 3.14 shows the fourth set of vortices being transferred to the pinning sites.

The program finishes after the fifth set of vortices has been transferred to the pinning sites. The completed simulation is shown in figure 3.15.

3.5 Building Up Persistent Current with Four Vortices

This simulation is an attempt to bring several vortices to a central pinning site in an attempt to see if vortices can exist in non-circular geometries. After four vortices have been transferred to a central pinning site, we place a blue-detuned laser in the center of the pinning site creating an annulus. The code for this simulation is available in appendix A.7.

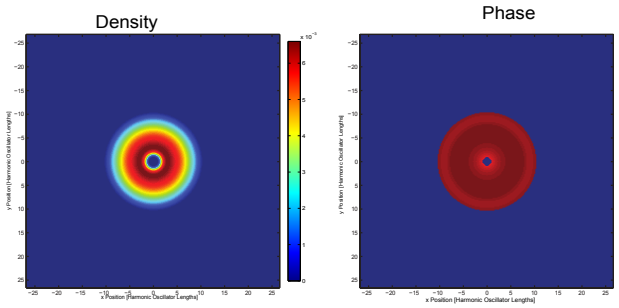

FIGURE 3.16: The central pinning site for the simulation

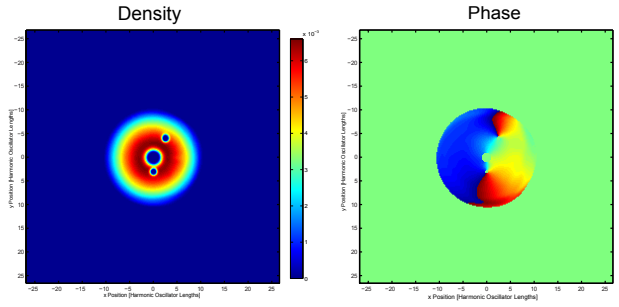

FIGURE 3.17: Pinning beam entering the central pinning site, dismissed beam headed towards the edge of the BEC

We allow the to BEC settle into its ground state as usual. From here we turn on a central pinning site. The site is larger than usual for reasons outlined section 3.4. 3.16 shows the central pinning site.

Because of the central pinning site, we chose to separate the beams in a circular path instead of a linear one. The beams seperate on a path that is at a constant radius with respect to the center of the BEC.

The central value of the beams ere rotated about the central pinning site by an angle of $\frac{\pi}{2}$, at which point, the pinning vortex goes directly into the central pinning site, and the dismissed vortex leaves the BEC. Figure 3.17 shows the change in trajectory of the

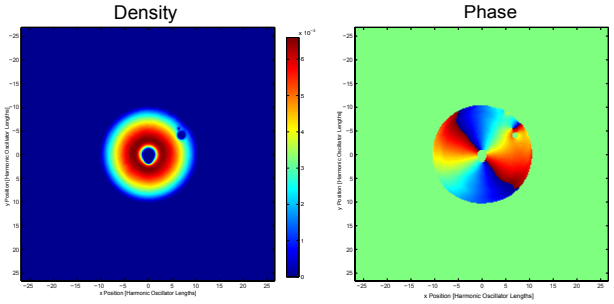

FIGURE 3.18: Second dismissed beam leaving the BEC creating edge vortices

beams.

As the dismissed beam leaves the BEC, it creates edge vortices which for the purpose of the simulation remain harmless and do not affect the primary objective of the simulation. Fig. 3.15 shows the second dismissed vortex beam leaving the BEC, while the edge vortex from the first beam has continued rotating autonomously around the edge.

This process is repeated four more times. As the vortex charge of the central pinning site increases, it becomes harder for the dismissed vortex to travel against the current. The vortex begins to get ripped from its beam. This limits the number of vortices that can be transferred to the central pinning site this way to about 4.

After we have transferred vortices to the central pinning site, we turn on ramp up the central beam. Figure 3.19 shows the central beam going through the central pinning site.

We see although the external blue-detuned laser is present, a central peak does not seem to exist.

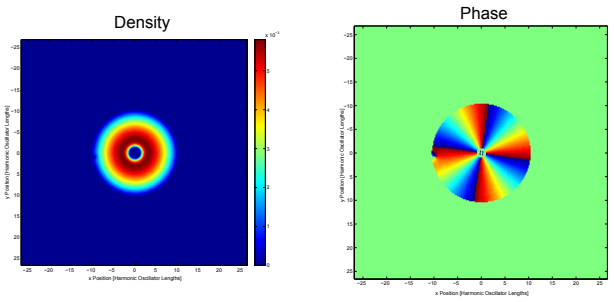

FIGURE 3.19: The central pinning site for the simulation

3.6 Dense Pinning Sites Containing One Type of Charge

The goal of the simulation is to push the limits of how many vortices of the same charge we could get into a BEC. We create a densely packed arrangement of pinning sites, that turn on one at a time. To get all of the pinning sites to have the same charge, we need to consistently remove one vortex for every pair that is created. The code for this simulation is available in appendix A.8.

As discussed in section 3.3, removing vortices from BECs can create issues, such as edge vortices or problems with vortex creation. To address these issues, this specific experiment utilizes something we refer to as an edge reservoir.

If we try to remove vortices by simply moving dismissed vortices one at a time, we see some of the issues discussed in section 3.3. The first attempt at solving this problem was to create a reservoir. Dismissed vortices, would be brought to a pinning site. When all of the pinned vortices were in their final location, then the reservoir of dismissed vortices would be brought out of the BEC.

This turned not to work out, the reservoir had trouble holding all of the dismissed vortices. After four DVs, the reservoir trap would start leaking free vortices if any more

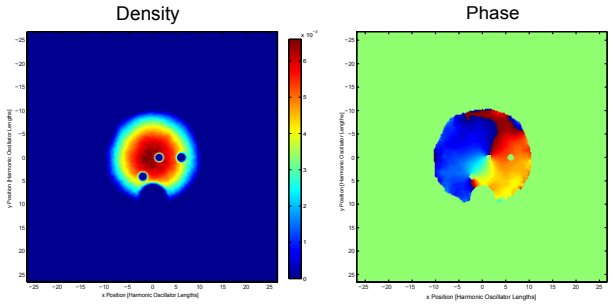

FIGURE 3.20: A vortex pair creation. The dismissed vortex is headed towards the edge reservoir

were transferred in (as seen in figure 3.11). As discussed in section 3.4, it is possible to change the trap depth or width to accommodate more vortices, but doing so eats up valuable BEC pinning site space. Essentially, the bigger the reservoir, the fewer pinning vortices sites that could be created.

The solution was the creation of the edge reservoir. This reservoir connects interior of the BEC with the edge. By transferring vortices to the edge reservoir, the dismissed vortex can be removed from the BEC without issue. Figure 3.20 demonstrates the pinning site, along with the edge reservoir at the bottom.

Due to the large number of pinning sites, it was better to generate each pinning site using a for loop than create a whole module for each pinning site creation and beam transfer. Each iteration of the for loop designates a specific pinning site location. This is accomplished through the use of Boolean logic operators. Essentially, a pinning site location is designated for each specific i value of the for loop. When the i value is equal a specific number, the Boolean operator is one, and zero for all other values. Each value is multiplied by the value of the individual position.

```
xposit= (i ==1)*6+(i ==2)*3+(i ==3)*3+(i ==4)*3+(i ==5)*1+(i ==6)*1+(i ==7)*1+(i ==8)*(-1);
yposit= (i ==1)*0+(i ==2)*3+(i ==3)*(-3)+(i ==4)*(0)+(i ==5)*(-5)+(i ==6)*(-2)+(i ==7)*(2)+(i ==8);
```

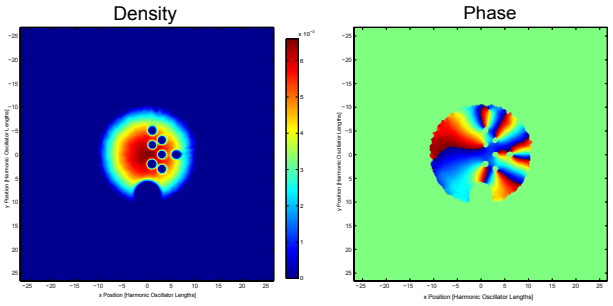

FIGURE 3.21: A vortex of the same charge in each of the pinning sites.

From each position, the trajectories and pinning sites are located. For the beam transfer section of the code, all positions need to be shifted by one forward. So that the new beam locations are generated as the old ones are being taken away. Also at the movemax value for each loop would change with each iteration. This allows us to customize how quickly each vortex is being created or destroyed.

```
movemax4 = (i ==1)*2500+(i ==2)*3000+(i ==3)*2500+(i ==4)*2250+(i ==5)*2000+(i ==6)*1500+(i ==7)*2250+(i ==8);
```

The program runs and moves each pinning vortex to its pinning site, and removes each dismissed vortex via the edge resivoir. Figure 3.21 shows all pinning sites being occupied by vortices of the same charge.

After all of the pinned vortices were transferred to their sites, we turned down the pinning site beams. This caused the beams to become free, and we could watch the dynamics of the vortices without any external influence. Figure 3.22 shows the free vortices roaming through the BEC.

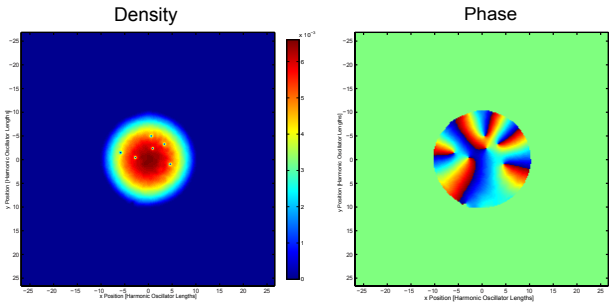

FIGURE 3.22: Free vortices after pinning sites were turned off.

3.7 Circlular Vortex Beam Generation Distribution

In section 3.5 we attempt to build up a persistent current around a central pinning site by transferring vortices in one at a time. As the current builds up, it become harder to transfer more vortices into the pinning site. In an attempt to generate a simulation where a large number of vortices could be quickly transferred to a central pinning site the circle dance program was generated. Although the program succeeds in doing this, some of the other results that are created are interesting to study. The code for this section available in appendix A.9.

The idea is that a central pinning site is created, then at some radius away, an arbitrary number beam generation sites were created, evenly spaced from each other. Then, the whole ring of generation beams begins rotating. One set of beams would continue to rotate at the fixed radius, while the other set of beams would decrease radius, while maintaining rotation so that they move to the central pinning site.

The program starts as usual with the settling of the ground state, and then turning down the damping. Once the number of generation beams is determined, the program turns on that number of beams at the specified radius. Figure 3.23 shows a configuration of

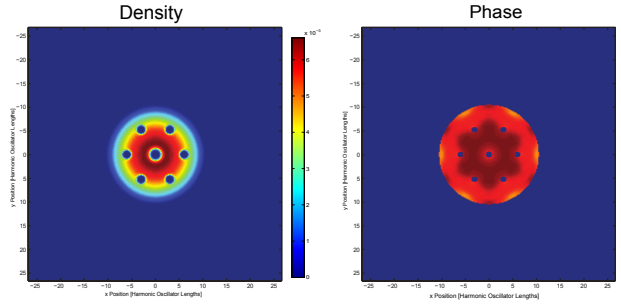

FIGURE 3.23: Six vortex generation sites at a radius of harmonic oscillator lengths, plus the central pinning site.

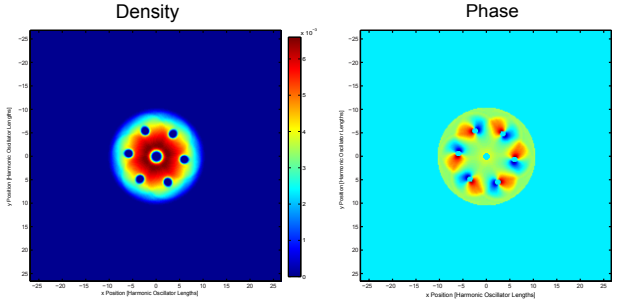

FIGURE 3.24: The vortex creation sites being rotated.

six generations sites equally spaced at a radius of six harmonic oscillator lengths, along with the central pinning site.

The beam generations sites are created after we determine the number of sites, and the radius that they are to be created at.

Once all of the beams are turned on, the ring of beam generation sites rotate at a constant angular speed. The radius of the pinning beams is decreased, while maintaining the same angular speed. The dismissed beams continue to rotate at a constant rate. The beam rotation is shown in figure 3.24. The pinning beams inspiraling is shown in figure 3.25.

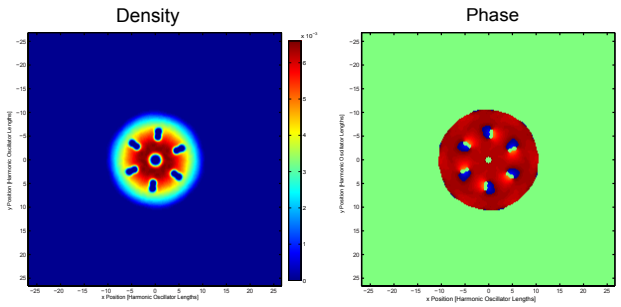

FIGURE 3.25: Pinning sites inspiraling.

Depending on the number of beam, and the radius that the beams are created at, we can generate two types of effects as the vortices are brought in.

If the number of generation beams is low enough, then the vortex beams headed toward the pinning site will not touch each other before they transfer their vortices the central beam. As they head inward they transfer their beams towards the central pinning site. This effectively transfers all vortices to the central pinning site. If we wish, we can move all of the dismissed vortices out of the BEC. Figures 3.26 and 3.27 show the pinning beams inspiraling, and the successful beam transfer.

This method turns out to be an effective method for placing a large number of vortices into a central pinning site, without any major issues, certainly more effective than what was attempted in section. 3.5.

The second possible case occurs, when a large number of beam generation sites is created. If we have a large enough number of beam generation sites, as the pinning beams move towards the pinning site, they touch each other before they transfer to the central pinning site. The individual pinning beams, then form an annulus. As the annulus is created, a geometry where vortices cannot exist is created, and all of the vortices in the pinning beams are ejected. This creates a set of free vortices.

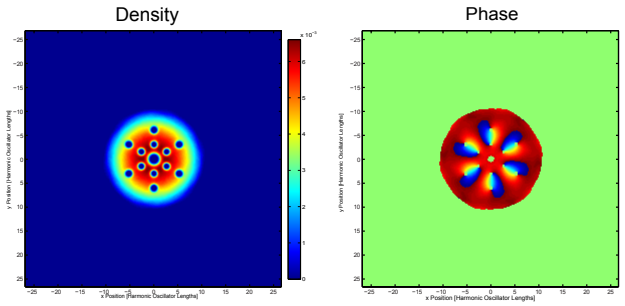

FIGURE 3.26: Pinning beams moving towards central pinning site.

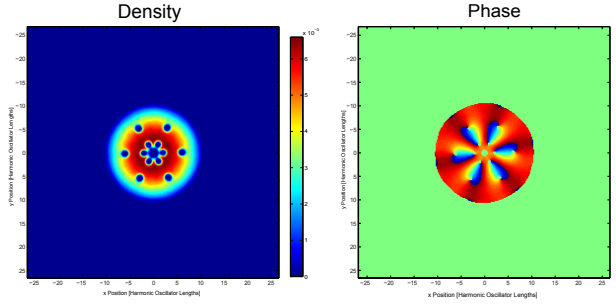

FIGURE 3.27: The six vortex beams being transferred to the central pinning site simultaneously.

The beams will intersect with each other before they interact with the central pinning site. This will create an annulus of beams, as they interact. The vortices from each beam are ejected and they become free vortices. Figures 3.28 and 3.29 show the annulus formation and vortex ejection.

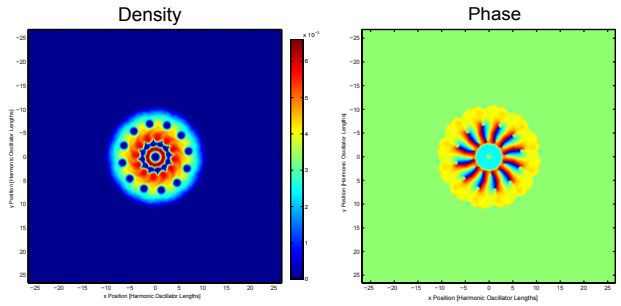

FIGURE 3.28: As the many vortex creation beams get closer to the central pinning site, they begin to form an Annulus. Vortices cannot stably exist under these conditions and begin leave their beams.

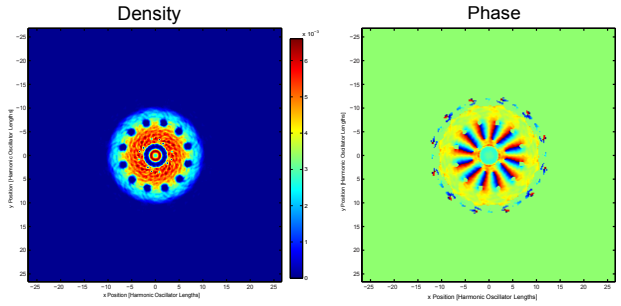

FIGURE 3.29: An annulus of vortex creation beam, with all of the vortices vacated from these beams.

Chapter 4

CONCLUSION

From section 3.2, we show how we can sucessfully create and transfer vortices. We can stably pin vortices to specific locations. Section 3.3 shows us how we can sucessfully remove unwanted vortices from a system, and some possible reprocussions of removing them improperly. We learned how we can store more than a single vortex in a specified pinnning site in section 3.4. From section 3.5 we learned that as the current in the system increased, our conditions for vortex creation will change. Addressing these changing conditions is the only way to continue creating new vortices. Section 3.6 showed the introduction of the edge reservoir and how we can address the problem of complex pinning site arrangements. Lastly section 3.7 showed us a method for placing several vortices into a central pinning site in a single round of vortex creation beams.

From the results section, we can see that it is possible to generate arbitrary vortex arrangements within a BEC. The next step for this work is to attempt them experimentally.

The problem lay in that whether or not these simulations are experimentally realizable. Simulations like that of sections 3.6 and 3.7, contain a large number of external

beams. Being able to stably control that number of beams within a BEC can be hard to experimentally realize.

Another factor to take into account is BEC lifetimes. Although our simulation can be run indefinitely, A simulation lasting longer than the stable lifetime of the BEC you are working on, will not be experimentally realizable.

Appendix A

An Appendix

A.1 BECShapeGen Code

```
function [BECINIT,sigx,g2D] = TwoDBECGenCorrect();


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function is responsible for generating the correct shape of a BEC
%based on the intial physical parameters.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N = 140;    %Number of HG polynimals created
mx = 2^8;   %Scale, must be a factor of 2^n so that it is compatible with FFT

%Calls the HG calculator

[H] = HermGaus(mx,N);

%Scale defined

x = linspace(-100,100,mx+1);
x = x(1:mx);
x = x*1E-6;
y = x;

sigx = x./sigmax;
sigy = sigx;
dx = -(sigx(1) - sigx(2));
dy = dx;
[Sigx,Sigy] = meshgrid(sigx,sigy);

%%%%%%%%%%%%%%%% PHYSICAL PARAMETERS %%%%%%%%%%%%%%

%Here we set Sigmax = Sigmay. If we want to change this (make an oblate
%BEC) here would be spot to change things.

%Trap Frequencies
sigmax = 3.74E-6;   %8 hz
sigmay = sigmax;
sigmaz = 3.5E-6;   %effective trap thickness due to interactions

scatlen = 5.5E-9;
NumPar = 1e6;

g2D_dimensionless = sqrt(8*pi)*scatlen/sigmaz;
g2D = NumPar * g2D_dimensionless;

%%%%%%%%%%%%%%%% BEC ROUGH SHAPE DEFINITION %%%%%%%%%%%%%%%%

```

```
E = sqrt(g2D);   %chemical potential in dimensionless units
CPT =  sqrt(2*E);

BECXY = sqrt(1-(Sigx/CPT).^2-(Sigy/CPT).^2); %Defines the shape of the BEC
BECXY = BECXY .* (BECXY > 0);

A = sqrt(sum(sum(abs(BECXY).^2)))*dx;
BECXY = BECXY/A;


%%% these two lines might just be equal to N, verify and check.
Ender = size(H);
Ender = (Ender(1));


%%%%%%%%%% COEFFICENT CALCULATION AND SMOOTH DEFINION %%%%%%%%%%%%%%%%

BECINIT = 0;
for(i = 1:Ender)
    for(j = 1:Ender-i)
    HERMGAUSNOW = transpose(H(i,:))*(H(j,:));
    Coef2 = sum(sum(HERMGAUSNOW .* BECXY*dx*dy));
    %Coef2 = sum(transpose(Coef1))
    BECINIT = BECINIT + Coef2 * HERMGAUSNOW;

    end

end


A = sqrt(sum(sum(abs(BECINIT).^2)))*dx; %Norm
BECINIT = BECINIT/A;



%plot(x,BECINIT(:,mx/2))
imagesc(x,y,BECINIT);
axis image
```

A.2 HermGaus.m

```
function [psi] = HermGaus(mx,order);

sigma = 1.1E-6;



x = linspace(-20*sigma,20*sigma,mx+1);
x = x(1:mx);

sigx = x./sigma;

H = [mx,order];
psi = [mx,order];



for(j=1:mx)

H(1,:) =  1;
H(2,j) =  2.*sigx(j);
H(1,mx) = 1;
n = 0;
psi(1,j) = 1/(sqrt(2.^n*factorial(n)))*1/(sqrt(sigma))*pi^(-1/4)*exp(-.5*x(j)^2./(sigma^2))*H(1,
n = 1;
psi(2,j) = 1/(sqrt(2.^n*factorial(n)))*1/(sqrt(sigma))*pi^(-1/4)*exp(-.5*x(j)^2./(sigma^2))*H(2,
```

```
27
28 end
29
30 d = H(2,mx);
31
32 for(i = 3:order)
33
34 n = i - 1;
35
36 for(k=1:mx)
37
38 H(n+1,k) = 2*sigx(k).*H(n,k) - 2*(n-1)*H(n-1,k);
39
40 psi(n+1,k) = 1/(sqrt(2.^n*factorial(n)))*1/(sqrt(sigma))*pi^(-1/4)*exp(-.5*x(k)^2./(sigma^2))*H(
41 end
42
43 end
```

A.3 BECBPMcode

```
1  %function [x] = BECBPMcode(BECInit,sigx,g2D);
2
3  % Initial Parameters
4
5  %clear all
6
7
8  [BECInit,sigx,g2D] = BECShapeGen();
9
10 N = length(sigx);
11
12 xmax = -2*min(sigx);
13
14 t = (2 * xmax^2)/(pi^2*N^2);
15 damp = 0.03;
16
17 %Movemax determines the number of loop iterations.
18 movemax = 2000;
19 tmax = movemax * t;
20 gmax = g2D;
21
22
23 % X scaling.
24 dx = xmax/N;
25 nmid = floor (N/2);
26 v0 = [0:N-1];
27 x = (v0 * dx) - (xmax/2);
28 y = x;
29 [X,Y] = meshgrid(x,y);
30
31 %k Scaling.
32 kmax = (2 * pi)/(dx);
33 dk = kmax/N;
34 p = find(v0 > nmid);
35 vp = v0;
36 vp(p) = (N - v0(p));
37 eta = vp * dk;
38 nu = eta;
39
40 [Eta,Nu] = meshgrid(eta,nu);
41
42 %First transform from psi to phi
43
44 Psi0 = BECInit;
45
46 %Intial conitions before loop.
47 PsiOut = Psi0;
```

```
ticker = 0;
%density1 = abs(PsiOut.*conj(PsiOut));



for(k =1:movemax);


%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;

i2 = 1i-damp;

Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);


%Transform to momentum space


Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);

%Transform back to free space

Psi2 = fft2(Phi0);


density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);



A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;



P2 = abs(PsiOut.*conj(PsiOut));
p02 = abs(Psi0.*conj(Psi0));
imagesc(x,y,P2);
%plot(x,P2)
drawnow

end
```

A.4 Code for 3.2

```
%function [x] = (BECInit,sigx,g2D);

% Initial Parameters

%clear all


[BECInit,sigx,g2D] = TwoDBECGenCorrect();

N = length(sigx);
sigmax = 3.74E-6;
xmax = -2*min(sigx);

t = 1*(2 * xmax^2)/(pi^2*N^2);
damp1= 0.03;
damp2= 0.003;
```

```
17
18
19 %Movemax determines the number of loop iterations.
20 movemax1 = 1000;
21 movemax2 = 100;
22 movemax3 = 1000;
23 movemax4 = 1000;
24 movemax5 = 500;
25 movemax6 = 600;
26
27
28 tmax = movemax1 * t;
29 gmax = g2D;
30 E = sqrt(g2D);
31
32 % X scaling.
33 dx = xmax/N;
34 nmid = floor (N/2);
35 v0 = [0:N-1];
36 x = (v0 * dx) - (xmax/2);
37 y = x;
38 [X,Y] = meshgrid(x,y);
39
40 %k Scaling.
41 kmax = (2 * pi)/(dx);
42 dk = kmax/N;
43 p = find(v0 > nmid);
44 vp = v0;
45 vp(p) = (N - v0(p));
46 eta = vp * dk;
47 nu = eta;
48
49 [Eta,Nu] = meshgrid(eta,nu);
50
51 %First transform from psi to phi
52
53 Psi0 = BECInit;
54
55 %Intial conitions before loop.
56 PsiOut = Psi0;
57 ticker = 0;
58
59
60 %STAGE 1: Stabilization of Ground state
61
62 for(k =1:movemax1);
63
64
65 %First Evolution step in free space
66
67 density = abs(PsiOut.*conj(PsiOut));
68 Interm = gmax * density;
69
70 i2 = 1i-damp1;
71
72 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
73
74 %Transform to momentum space
75
76
77 Phi0=ifft2(Psi1);
78 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
79 %ticker = ticker + t/2;
80
81 %Transform back to free space
82
83
84
85 Psi2 = fft2(Phi0);
86
87
```

```
density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
%ticker = ticker + t/4;


A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;



P2 = abs(PsiOut.*conj(PsiOut));
p02 = abs(Psi0.*conj(Psi0));
%Graphing

imagesc(x,y,P2);

xlabel('x Position [microns]');
ylabel('y Position [microns]');
title('Density')

axis image;
drawnow


end

%STAGE 3: Turn down Damping

dampticker = 0;
for(k =1:movemax2);

dampticker = dampticker + 1;

%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;

dampdown = .03 - ((.03-.003)/movemax2)*dampticker;

i2 = 1i-dampdown;

Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);

%Transform to momentum space


Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
%ticker = ticker + t/2;

%Transform back to free space

%Psi2 = fftshift(ifft(fftshift(Phi0)));

Psi2 = fft2(Phi0);


density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);


A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;


```

```
159 P2 = abs(PsiOut.*conj(PsiOut));
160
161 %Graphing Area
162
163 subplot(1,2,1)
164 xlabel('x Position [microns]');
165 ylabel('y Position [microns]');
166 title('Density')
167 imagesc(x,y,P2);
168 axis image
169 subplot(1,2,2)
170 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
171 imagesc(x,y,anglef);
172 xlabel('x Position [microns]');
173 ylabel('y Position [microns]');
174 title('Phase');
175 axis image
176 drawnow
177
178 end
179
180
181
182 %Generation of third and fourth Laser
183
184 ticker = 0;
185 for(k =1:movemax5);
186
187     x3t = 2;
188     y3t = 2;
189     x4t = 2;
190     y4t = -2;
191     x5t = 0;
192     y5t = 4;
193     x6t = 0;
194     y6t = -4;
195
196 Vmax = 1.2*E;
197 dtild = .8;
198
199 Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtild^2))*ticker/movemax5;
200 Vlaser4 = Vmax*exp(-2*((X-x4t).^2+(Y-y4t).^2)/(dtild^2))*ticker/movemax5;
201 Vlaser5 = Vmax*exp(-2*((X-x5t).^2+(Y-y5t).^2)/(dtild^2))*ticker/movemax5;
202 Vlaser6 = Vmax*exp(-2*((X-x6t).^2+(Y-y6t).^2)/(dtild^2))*ticker/movemax5;
203
204 Vlasers = Vlaser3+Vlaser4+Vlaser5+Vlaser6;
205 ticker = ticker + 1;
206
207 %First Evolution step in free space
208
209 density = abs(PsiOut.*conj(PsiOut));
210 Interm = gmax * density;
211
212
213 i2 = 1i-damp2;
214 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
215
216 %Transform to momentum space
217
218 %Phi0 = fftshift(fft(fftshift(Psi1)));
219 Phi0=ifft2(Psi1);
220 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
221
222 %Transform back to free space
223
224 %Psi2 = fftshift(ifft(fftshift(Phi0)));
225
226 Psi2 = fft2(Phi0);
227
228
229 density = abs(Psi2.*conj(Psi2));
```

```
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);



A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;



P2 = abs(PsiOut.*conj(PsiOut));
p02 = abs(Psi0.*conj(Psi0));

%Graphing Area

subplot(1,2,1)
xlabel('x Position [microns]');
ylabel('y Position [microns]');
title('Density')
imagesc(x,y,P2);
axis image
subplot(1,2,2)
anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
imagesc(x,y,anglef);
xlabel('x Position [microns]');
ylabel('y Position [microns]');
title('Phase');
axis image
drawnow


end



%STAGE 2: Generation of Laser

ticker = 0;
for(k =1:movemax3);

    x1t = -5;
    y1t = 0;
    x2t = -5;
    y2t = 0;
    x3t = 2;
    y3t = 2;
    x4t = 2;
    y4t = -2;
    x5t = 0;
    y5t = 4;
    x6t = 0;
    y6t = -4;

Vmax = 1.2*E;
dtild = .8;

Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2))*ticker/movemax3;
Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2))*ticker/movemax3;
Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtild^2));
Vlaser4 = Vmax*exp(-2*((X-x4t).^2+(Y-y4t).^2)/(dtild^2));
Vlaser5 = Vmax*exp(-2*((X-x5t).^2+(Y-y5t).^2)/(dtild^2));
Vlaser6 = Vmax*exp(-2*((X-x6t).^2+(Y-y6t).^2)/(dtild^2));


ticker = ticker + 1;

%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;
```

```
301
302
303 i2 = 1i-damp1;
304 Vlasers = Vlaser1 +Vlaser2+Vlaser3+Vlaser4+Vlaser5+Vlaser6;
305 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlasers).*t./i2/2);
306
307 %Transform to momentum space
308
309 %Phi0 = fftshift(fft(fftshift(Psi1)));
310 Phi0=ifft2(Psi1);
311 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
312
313 %Transform back to free space
314
315
316 Psi2 = fft2(Phi0);
317
318
319 density = abs(Psi2.*conj(Psi2));
320 Interm = gmax *density;
321
322 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
323
324
325
326 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
327 PsiOut = PsiOut/A;
328
329
330
331 P2 = abs(PsiOut.*conj(PsiOut));
332
333 %Graphing Area
334
335 subplot(1,2,1)
336 xlabel('x Position [microns]');
337 ylabel('y Position [microns]');
338 title('Density')
339 imagesc(x,y,P2);
340 axis image
341 subplot(1,2,2)
342 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
343 imagesc(x,y,anglef);
344 xlabel('x Position [microns]');
345 ylabel('y Position [microns]');
346 title('Phase');
347 axis image
348 drawnow
349
350
351 end
352
353
354
355
356 ticker = 0;
357 for(k =1:movemax4);
358
359
360
361     trot = ticker/movemax4;
362
363     x3t = 2;
364     y3t = 2;
365     x4t = 2;
366     y4t = -2;
367     x5t = 0;
368     y5t = 4;
369     x6t = 0;
370     y6t = -4;
371
```

```
372
373     %P1
374
375
376     x1t = -5+7*trot;
377     y1t = 2*trot;
378
379     %P2
380
381     x2t = -5+7*trot;
382     y2t = -2*trot;
383
384 Vmax = 1.2*E;
385 dtild = .8;
386 Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2));
387 Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2));
388 Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtild^2));
389 Vlaser4 = Vmax*exp(-2*((X-x4t).^2+(Y-y4t).^2)/(dtild^2));
390 Vlaser5 = Vmax*exp(-2*((X-x5t).^2+(Y-y5t).^2)/(dtild^2));
391 Vlaser6 = Vmax*exp(-2*((X-x6t).^2+(Y-y6t).^2)/(dtild^2));
392
393
394 ticker = ticker + 1;
395
396 %First Evolution step in free space
397
398 density = abs(PsiOut.*conj(PsiOut));
399 Interm = gmax * density;
400
401
402 i2 = 1i-damp2;
403 Vlasers = Vlaser1 +Vlaser2+Vlaser3+Vlaser4+Vlaser5+Vlaser6;
404 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlasers).*t./i2/2);
405
406 %Transform to momentum space
407
408 %Phi0 = fftshift(fft(fftshift(Psi1)));
409 Phi0=ifft2(Psi1);
410 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
411
412 %Transform back to free space
413
414 Psi2 = fft2(Phi0);
415
416
417 density = abs(Psi2.*conj(Psi2));
418 Interm = gmax *density;
419
420 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
421 %ticker = ticker + t/4;
422
423
424 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
425 PsiOut = PsiOut/A;
426
427
428
429
430 P2 = abs(PsiOut.*conj(PsiOut));
431 p02 = abs(Psi0.*conj(Psi0));
432
433 %Graphing Area
434
435 subplot(1,2,1)
436 xlabel('x Position [microns]');
437 ylabel('y Position [microns]');
438 title('Density')
439 imagesc(x,y,P2);
440 axis image
441 subplot(1,2,2)
442 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
```

```
imagesc(x,y,anglef);
xlabel('x Position [microns]');
ylabel('y Position [microns]');
title('Phase');
axis image
drawnow


end

ticker = 0;
for(k =1:movemax3);

    x1t = -5;
    y1t = 0;
    x2t = -5;
    y2t = 0;
    x3t = 2;
    y3t = 2;
    x4t = 2;
    y4t = -2;
    x5t = 0;
    y5t = 4;
    x6t = 0;
    y6t = -4;

Vmax = 1.2*E;
dtild = .8;

Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2))*ticker/movemax3;
Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2))*ticker/movemax3;
Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtild^2));
Vlaser4 = Vmax*exp(-2*((X-x4t).^2+(Y-y4t).^2)/(dtild^2));
Vlaser5 = Vmax*exp(-2*((X-x5t).^2+(Y-y5t).^2)/(dtild^2));
Vlaser6 = Vmax*exp(-2*((X-x6t).^2+(Y-y6t).^2)/(dtild^2));


ticker = ticker + 1;

%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;


i2 = 1i-damp1;
Vlasers = Vlaser1 +Vlaser2+Vlaser3+Vlaser4+Vlaser5+Vlaser6;
Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlasers).*t./i2/2);
%Transform to momentum space

Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);

%Transform back to free space

Psi2 = fft2(Phi0);


density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);


A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;



P2 = abs(PsiOut.*conj(PsiOut));

```

```
514 subplot(1,2,1)
515 xlabel('x Position [microns]');
516 ylabel('y Position [microns]');
517 title('Density')
518 imagesc(x,y,P2);
519 axis image
520 subplot(1,2,2)
521 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
522 imagesc(x,y,anglef);
523 xlabel('x Position [microns]');
524 ylabel('y Position [microns]');
525 title('Phase');
526 axis image
527 drawnow
528
529 end
530
531 ticker = 0;
532 for(k =1:movemax6);
533
534
535
536     trot = ticker/movemax6;
537
538     x3t = 2;
539     y3t = 2;
540     x4t = 2;
541     y4t = -2;
542     x5t = 0;
543     y5t = 4;
544     x6t = 0;
545     y6t = -4;
546
547
548     %P1
549
550
551     x1t = -5+5*trot;
552     y1t = 4*trot;
553
554     %P2
555
556     x2t = -5+5*trot;
557     y2t = -4*trot;
558
559 Vmax = 1.2*E;
560 dtild = .8;
561 Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2));
562 Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2));
563 Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtild^2));
564 Vlaser4 = Vmax*exp(-2*((X-x4t).^2+(Y-y4t).^2)/(dtild^2));
565 Vlaser5 = Vmax*exp(-2*((X-x5t).^2+(Y-y5t).^2)/(dtild^2));
566 Vlaser6 = Vmax*exp(-2*((X-x6t).^2+(Y-y6t).^2)/(dtild^2));
567
568
569 ticker = ticker + 1;
570
571 %First Evolution step in free space
572
573 density = abs(PsiOut.*conj(PsiOut));
574 Interm = gmax * density;
575
576
577 i2 = 1i-damp2;
578 Vlasers = Vlaser1 +Vlaser2+Vlaser3+Vlaser4+Vlaser5+Vlaser6;
579 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlasers).*t./i2/2);
580
581 %Transform to momentum space
582
583 Phi0=ifft2(Psi1);
584 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
```

```
585
586 %Transform back to free space
587
588 Psi2 = fft2(Phi0);
589
590
591 density = abs(Psi2.*conj(Psi2));
592 Interm = gmax *density;
593
594 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
595
596
597 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
598 PsiOut = PsiOut/A;
599
600
601
602
603 P2 = abs(PsiOut.*conj(PsiOut));
604 p02 = abs(Psi0.*conj(Psi0));
605
606 %Graphing Area
607
608 subplot(1,2,1)
609 xlabel('x Position [microns]');
610 ylabel('y Position [microns]');
611 title('Density')
612 imagesc(x,y,P2);
613 axis image
614 subplot(1,2,2)
615 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
616 imagesc(x,y,anglef);
617 xlabel('x Position [microns]');
618 ylabel('y Position [microns]');
619 title('Phase');
620 axis image
621 drawnow
622
623
624 end
```

A.5 Code for 3.3

```
1  %function [x] = (BECInit,sigx,g2D);
2
3  % Initial Parameters
4
5  %clear all
6
7
8  [BECInit,sigx,g2D] = TwoDBECGenCorrect();
9
10 N = length(sigx);
11 sigmax = 3.74E-6;
12 xmax = -2*min(sigx);
13
14 t = 1*(2 * xmax^2)/(pi^2*N^2);
15 damp1= 0.03;
16 damp2= 0.003;
17
18
19 %Movemax determines the number of loop iterations.
20 movemax1 = 1000;
21 movemax2 = 100;
22 movemax3 = 1000;
23 movemax4 = 1056;
24 movemax5 = 500;
```

```
movemax6 = 600;
movemax7 = 200;


tmax = movemax1 * t;
gmax = g2D;
E = sqrt(g2D);

% X scaling.
dx = xmax/N;
nmid = floor (N/2);
v0 = [0:N-1];
x = (v0 * dx) - (xmax/2);
y = x;
[X,Y] = meshgrid(x,y);

%k Scaling.
kmax = (2 * pi)/(dx);
dk = kmax/N;
p = find(v0 > nmid);
vp = v0;
vp(p) = (N - v0(p));
eta = vp * dk;
nu = eta;

[Eta,Nu] = meshgrid(eta,nu);

%First transform from psi to phi

Psi0 = BECInit;

%Intial conitions before loop.
PsiOut = Psi0;
ticker = 0;


%STAGE 1: Stabilization of Ground state

for(k =1:movemax1);


%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;

i2 = 1i-damp1;

Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);

%Transform to momentum space


Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
%ticker = ticker + t/2;

%Transform back to free space



Psi2 = fft2(Phi0);


density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
%ticker = ticker + t/4;


```

```
A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;



P2 = abs(PsiOut.*conj(PsiOut));
p02 = abs(Psi0.*conj(Psi0));
imagesc(x,y,P2);
axis image;
drawnow



end

%STAGE 3: Turn down Damping

dampticker = 0;
for(k =1:movemax2);

dampticker = dampticker + 1;

%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;

dampdown = .03 - ((.03-.003)/movemax2)*dampticker;

i2 = 1i-dampdown;

Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);

%Transform to momentum space


Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
%ticker = ticker + t/2;

%Transform back to free space

%Psi2 = fftshift(ifft(fftshift(Phi0)));

Psi2 = fft2(Phi0);


density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);


A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;



P2 = abs(PsiOut.*conj(PsiOut));

subplot(1,2,1)
imagesc(x,y,P2);
axis image
subplot(1,2,2)
anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
imagesc(x,y,anglef);
axis image
p02 = abs(Psi0.*conj(Psi0));
drawnow
end

```

```
167
168
169 %Generation of third and fourth Laser
170
171 ticker = 0;
172 for(k =1:movemax5);
173
174     x3t = 2;
175     y3t = 2;
176
177
178 Vmax = 1.2*E;
179 dtild = .8;
180
181 Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtild^2))*ticker/movemax5;
182
183
184 Vlasers = Vlaser3;
185 ticker = ticker + 1;
186
187 %First Evolution step in free space
188
189 density = abs(PsiOut.*conj(PsiOut));
190 Interm = gmax * density;
191
192
193 i2 = 1i-damp2;
194 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
195
196 %Transform to momentum space
197
198 %Phi0 = fftshift(fft(fftshift(Psi1)));
199 Phi0=ifft2(Psi1);
200 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
201
202 %Transform back to free space
203
204 %Psi2 = fftshift(ifft(fftshift(Phi0)));
205
206 Psi2 = fft2(Phi0);
207
208
209 density = abs(Psi2.*conj(Psi2));
210 Interm = gmax *density;
211
212 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
213
214
215
216 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
217 PsiOut = PsiOut/A;
218
219
220
221 P2 = abs(PsiOut.*conj(PsiOut));
222 p02 = abs(Psi0.*conj(Psi0));
223
224 subplot(1,2,1)
225 imagesc(x,y,P2);
226 axis image
227 subplot(1,2,2)
228 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
229 imagesc(x,y,anglef);
230 axis image
231
232 drawnow
233 end
234
235
236
237 %STAGE 2: Generation of Laser
```

```
238
239 ticker = 0;
240 for(k =1:movemax3);
241
242     x1t = -5;
243     y1t = 0;
244     x2t = -5;
245     y2t = 0;
246     x3t = 2;
247     y3t = 2;
248
249
250 Vmax = 1.2*E;
251 dtild = .8;
252
253 Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2))*ticker/movemax3;
254 Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2))*ticker/movemax3;
255 Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtild^2));
256
257
258
259 ticker = ticker + 1;
260
261 %First Evolution step in free space
262
263 density = abs(PsiOut.*conj(PsiOut));
264 Interm = gmax * density;
265
266
267 i2 = 1i-damp1;
268
269 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlaser1 +Vlaser2+Vlaser3).*t./i2/2);
270
271 %Transform to momentum space
272
273 %Phi0 = fftshift(fft(fftshift(Psi1)));
274 Phi0=ifft2(Psi1);
275 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
276
277 %Transform back to free space
278
279
280 Psi2 = fft2(Phi0);
281
282
283 density = abs(Psi2.*conj(Psi2));
284 Interm = gmax *density;
285
286 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlaser1 + Vlaser2+Vlaser3).*t./i2/2);
287
288
289
290 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
291 PsiOut = PsiOut/A;
292
293
294
295 P2 = abs(PsiOut.*conj(PsiOut));
296
297 subplot(1,2,1);
298 imagesc(x,y,P2);
299 axis image
300 subplot(1,2,2);
301 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
302 imagesc(x,y,anglef);
303 axis image
304 drawnow
305 end
306
307
308
```

```
309
310 ticker = 0;
311 for(k =1:movemax4);
312
313
314
315     trot = ticker/movemax4;
316
317     x3t = 2;
318     y3t = 2;
319
320
321     %P1
322
323
324     x1t = -5+7*trot;
325     y1t = 2*trot;
326
327     %P2
328
329     x2t = -5+7*trot;
330     y2t = -2*trot;
331
332 Vmax = 1.2*E;
333 dtild = .8;
334 Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2));
335 Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2));
336 Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtild^2));
337
338 ticker = ticker + 1;
339
340 %First Evolution step in free space
341
342 density = abs(PsiOut.*conj(PsiOut));
343 Interm = gmax * density;
344
345
346 i2 = 1i-damp2;
347 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlaser1 +Vlaser2+Vlaser3).*t./i2/2);
348
349 %Transform to momentum space
350
351 %Phi0 = fftshift(fft(fftshift(Psi1)));
352 Phi0=ifft2(Psi1);
353 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
354
355 %Transform back to free space
356
357 Psi2 = fft2(Phi0);
358
359
360 density = abs(Psi2.*conj(Psi2));
361 Interm = gmax *density;
362
363 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlaser1 + Vlaser2 + Vlaser3).*t./i2/2);
364 %ticker = ticker + t/4;
365
366
367 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
368 PsiOut = PsiOut/A;
369
370
371
372
373 P2 = abs(PsiOut.*conj(PsiOut));
374 p02 = abs(Psi0.*conj(Psi0));
375
376 subplot(1,2,1);
377 imagesc(x,y,P2);
378 axis image
379 subplot(1,2,2);
```

```
380
381 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
382 imagesc(x,y,anglef);
383 axis image
384 drawnow
385 end
386
387
388 ticker = 0;
389 for(k =1:movemax4);
390
391
392
393     trot = ticker/movemax4;
394
395     x3t = 2;
396     y3t = 2;
397
398
399     %P1
400
401
402     x1t = 2;
403     y1t = 2;
404
405     %P2
406
407     x2t = 2;
408     y2t = -2-9*trot;
409
410 Vmax = 1.2*E;
411 dtild = .8;
412 Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2));
413 Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2));
414 Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtild^2));
415
416 ticker = ticker + 1;
417
418 %First Evolution step in free space
419
420 density = abs(PsiOut.*conj(PsiOut));
421 Interm = gmax * density;
422
423
424 i2 = 1i-damp2;
425 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlaser1 +Vlaser2+Vlaser3).*t./i2/2);
426
427 %Transform to momentum space
428
429 %Phi0 = fftshift(fft(fftshift(Psi1)));
430 Phi0=ifft2(Psi1);
431 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
432
433 %Transform back to free space
434
435 Psi2 = fft2(Phi0);
436
437
438 density = abs(Psi2.*conj(Psi2));
439 Interm = gmax *density;
440
441 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlaser1 + Vlaser2 + Vlaser3).*t./i2/2);
442 %ticker = ticker + t/4;
443
444
445 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
446 PsiOut = PsiOut/A;
447
448
449
450
```

```
P2 = abs(PsiOut.*conj(PsiOut));
p02 = abs(Psi0.*conj(Psi0));

subplot(1,2,1);
imagesc(x,y,P2);
axis image
subplot(1,2,2);

anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
imagesc(x,y,anglef);
axis image
drawnow
end
```

A.6 Code for 3.4

```
%function [x] = (BECInit,sigx,g2D);

% Initial Parameters

%clear all


[BECInit,sigx,g2D] = TwoDBECGenCorrect();

N = length(sigx);
sigmax = 3.74E-6;
xmax = -2*min(sigx);
%xmax = 40/1.1;   %in scaled units
t = 1*(2 * xmax^2)/(pi^2*N^2);
damp1= 0.03;
damp2= 0.003;
%indexnum = [1:256];

%Movemax determines the number of loop iterations.
movemax1 = 1000;
movemax2 = 100;
movemax3 = 750;
movemax4 = 600;
movemax5 = 500;
movemax6 = 1250;


tmax = movemax1 * t;
gmax = g2D;
E = sqrt(g2D);

% X scaling.
dx = xmax/N;
nmid = floor (N/2);
v0 = [0:N-1];
x = (v0 * dx) - (xmax/2);
y = x;
[X,Y] = meshgrid(x,y);

%k Scaling.
kmax = (2 * pi)/(dx);
dk = kmax/N;
p = find(v0 > nmid);
vp = v0;
vp(p) = (N - v0(p));
eta = vp * dk;
nu = eta;

[Eta,Nu] = meshgrid(eta,nu);

%First transform from psi to phi
```

```
52
53 Psi0 = BECInit;
54
55 %Intial conitions before loop.
56 PsiOut = Psi0;
57
58
59
60 %STAGE 1: Stabilization of Ground state
61
62 for(k =1:movemax1);
63
64
65 %First Evolution step in free space
66
67 density = abs(PsiOut.*conj(PsiOut));
68 Interm = gmax * density;
69
70 i2 = 1i-damp1;
71
72 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
73
74 %Transform to momentum space
75
76 %Phi0 = fftshift(fft(fftshift(Psi1)));
77 Phi0=ifft2(Psi1);
78 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
79
80 %Transform back to free space
81
82 %Psi2 = fftshift(ifft(fftshift(Phi0)));
83
84 Psi2 = fft2(Phi0);
85
86
87 density = abs(Psi2.*conj(Psi2));
88 Interm = gmax *density;
89
90 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
91
92
93
94 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
95 PsiOut = PsiOut/A;
96
97
98
99 P2 = abs(PsiOut.*conj(PsiOut));
100
101
102 p02 = abs(Psi0.*conj(Psi0));
103
104 %Graphing Area
105
106 subplot(1,2,1)
107 xlabel('x Position [microns]');
108 ylabel('y Position [microns]');
109 title('Density')
110 imagesc(x,y,P2);
111 axis image
112 subplot(1,2,2)
113 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
114 imagesc(x,y,anglef);
115 xlabel('x Position [microns]');
116 ylabel('y Position [microns]');
117 title('Phase');
118 axis image
119 drawnow
120
121
122
```

```
123 end
124
125
126 %STAGE 3: Turn down Damping
127
128 dampticker = 0;
129 for(k =1:movemax2);
130
131 dampticker = dampticker + 1;
132
133 %First Evolution step in free space
134
135 density = abs(PsiOut.*conj(PsiOut));
136 Interm = gmax * density;
137
138 dampdown = .03 - ((.03-.003)/movemax2)*dampticker;
139
140 i2 = 1i-dampdown;
141
142 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
143
144 %Transform to momentum space
145
146
147 Phi0=ifft2(Psi1);
148 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
149
150
151 %Transform back to free space
152
153
154 Psi2 = fft2(Phi0);
155
156
157 density = abs(Psi2.*conj(Psi2));
158 Interm = gmax *density;
159
160 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
161
162
163 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
164 PsiOut = PsiOut/A;
165
166
167
168 P2 = abs(PsiOut.*conj(PsiOut));
169
170
171 %Graphing Area
172
173 subplot(1,2,1)
174 xlabel('x Position [microns]');
175 ylabel('y Position [microns]');
176 title('Density')
177 imagesc(x,y,P2);
178 axis image
179 subplot(1,2,2)
180 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
181 imagesc(x,y,anglef);
182 xlabel('x Position [microns]');
183 ylabel('y Position [microns]');
184 title('Phase');
185 axis image
186 drawnow
187
188 p02 = abs(Psi0.*conj(Psi0));
189 %plot(x,P2)
190 %drawnow
191 end
192
193
```

```
194
195 %Generation of third and fourth Laser
196
197 ticker = 0;
198 for(k =1:movemax5);
199
200     x3t = 2;
201     y3t = 3;
202     x4t = 2;
203     y4t = -3;
204
205 Vmaxs = 3.6*E;
206 dtilds = 1.4;
207
208 Vlaser3 = Vmaxs*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtilds^2))*ticker/movemax5;
209 Vlaser4 = Vmaxs*exp(-2*((X-x4t).^2+(Y-y4t).^2)/(dtilds^2))*ticker/movemax5;
210
211 ticker = ticker + 1;
212
213 %First Evolution step in free space
214
215 density = abs(PsiOut.*conj(PsiOut));
216 Interm = gmax * density;
217
218
219 i2 = 1i-damp2;
220
221 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm+Vlaser3+Vlaser4).*t./i2/2);
222 %ticker = ticker + t/4;
223 %Transform to momentum space
224
225 %Phi0 = fftshift(fft(fftshift(Psi1)));
226 Phi0=ifft2(Psi1);
227 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
228 %ticker = ticker + t/2;
229
230 %Transform back to free space
231
232 %Psi2 = fftshift(ifft(fftshift(Phi0)));
233
234 Psi2 = fft2(Phi0);
235
236
237 density = abs(Psi2.*conj(Psi2));
238 Interm = gmax *density;
239
240 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlaser3+Vlaser4).*t./i2/2);
241 %ticker = ticker + t/4;
242
243
244 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
245 PsiOut = PsiOut/A;
246
247
248
249 P2 = abs(PsiOut.*conj(PsiOut));
250 p02 = abs(Psi0.*conj(Psi0));
251
252 %Graphing Area
253
254 subplot(1,2,1)
255 xlabel('x Position [microns]');
256 ylabel('y Position [microns]');
257 title('Density')
258 imagesc(x,y,P2);
259 axis image
260 subplot(1,2,2)
261 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
262 imagesc(x,y,anglef);
263 xlabel('x Position [microns]');
264 ylabel('y Position [microns]');
```

```
title('Phase');
axis image
drawnow

drawnow
end



%Stage 4: Moving the beams.

for(m = 1:5)

%STAGE 2: Generation of Laser

ticker = 0;
for(k =1:movemax3);

    x1t = -5;
    y1t = 0;
    x2t = -5;
    y2t = 0;
    x3t = 2;
    y3t = 3;
    x4t = 2;
    y4t = -3;

Vmax = 1.2*E;
dtild = .8;

Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2))*ticker/movemax3;
Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2))*ticker/movemax3;
Vlaser3 = Vmaxs*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtilds^2));
Vlaser4 = Vmaxs*exp(-2*((X-x4t).^2+(Y-y4t).^2)/(dtilds^2));

ticker = ticker + 1;

%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;


i2 = 1i-damp1;

Vlasers = Vlaser1 +Vlaser2+Vlaser3+Vlaser4;
Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlasers).*t./i2/2);

%Transform to momentum space

%Phi0 = fftshift(fft(fftshift(Psi1)));
Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
%ticker = ticker + t/2;

%Transform back to free space

%Psi2 = fftshift(ifft(fftshift(Phi0)));

Psi2 = fft2(Phi0);


density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
%ticker = ticker + t/4;


A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;
```

```
336
337
338
339 P2 = abs(PsiOut.*conj(PsiOut));
340
341 %Graphing Area
342
343 subplot(1,2,1)
344 xlabel('x Position [microns]');
345 ylabel('y Position [microns]');
346 title('Density')
347 imagesc(x,y,P2);
348 axis image
349 subplot(1,2,2)
350 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
351 imagesc(x,y,anglef);
352 xlabel('x Position [microns]');
353 ylabel('y Position [microns]');
354 title('Phase');
355 axis image
356 drawnow
357
358 drawnow
359 end
360
361
362
363
364 ticker = 0;
365 for(k =1:movemax4);
366
367
368
369     trot = ticker/movemax4;
370
371     x3t = 2;
372     y3t = 3;
373     x4t = 2;
374     y4t = -3;
375
376
377     %P1
378
379
380     x1t = -5+7*trot;
381     y1t = 3*trot;
382
383     %P2
384
385     x2t = -5+7*trot;
386     y2t = -3*trot;
387
388 Vmax = 1.2*E;
389 dtild = .8;
390 Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2));
391 Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2));
392 Vlaser3 = Vmaxs*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtilds^2));
393 Vlaser4 = Vmaxs*exp(-2*((X-x4t).^2+(Y-y4t).^2)/(dtilds^2));
394 Vlasers = Vlaser1 +Vlaser2+Vlaser3+Vlaser4;
395 ticker = ticker + 1;
396
397 %First Evolution step in free space
398
399 density = abs(PsiOut.*conj(PsiOut));
400 Interm = gmax * density;
401
402
403 i2 = 1i-damp2;
404
405 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlasers).*t./i2/2);
406
```

```
%Transform to momentum space


Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);


%Transform back to free space


Psi2 = fft2(Phi0);


density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
%ticker = ticker + t/4;


A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;




P2 = abs(PsiOut.*conj(PsiOut));
p02 = abs(Psi0.*conj(Psi0));

%Graphing Area

subplot(1,2,1)
xlabel('x Position [microns]');
ylabel('y Position [microns]');
title('Density')
imagesc(x,y,P2);
axis image
subplot(1,2,2)
anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
imagesc(x,y,anglef);
xlabel('x Position [microns]');
ylabel('y Position [microns]');
title('Phase');
axis image
drawnow



%plot(x,P2)
drawnow
end
end
```

A.7 Code for 3.5

```
%function [x] = (BECInit,sigx,g2D);

% Initial Parameters

%clear all


[BECInit,sigx,g2D] = TwoDBECGenCorrect();

N = length(sigx);
sigmax = 3.74E-6;
xmax = -2*min(sigx);
```

```
%xmax = 40/1.1;   %in scaled units
t = 1*(2 * xmax^2)/(pi^2*N^2);
damp1= 0.03;
damp2= 0.003;
%indexnum = [1:256];

%Movemax determines the number of loop iterations.
movemax1 = 1000;
movemax2 = 100;
movemax3 = 1000;
movemax4 = 800;
movemax5 = 750;
movemax6 = 750;
movemax7 = 1000;
movemax8 = 1000;

tmax = movemax1 * t;
gmax = g2D;
E = sqrt(g2D);

% X scaling.
dx = xmax/N;
nmid = floor (N/2);
v0 = [0:N-1];
x = (v0 * dx) - (xmax/2);
y = x;
[X,Y] = meshgrid(x,y);

%k Scaling.
kmax = (2 * pi)/(dx);
dk = kmax/N;
p = find(v0 > nmid);
vp = v0;
vp(p) = (N - v0(p));
eta = vp * dk;
nu = eta;

[Eta,Nu] = meshgrid(eta,nu);

%First transform from psi to phi

Psi0 = BECInit;

%Intial conitions before loop.
PsiOut = Psi0;
ticker = 0;
%density1 = abs(PsiOut.*conj(PsiOut));

%STAGE 1: Stabilization of Ground state

for(k =1:movemax1);


%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;

i2 = 1i-damp1;

Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
%ticker = ticker + t/4;
%Transform to momentum space

%Phi0 = fftshift(fft(fftshift(Psi1)));
Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
%ticker = ticker + t/2;

%Transform back to free space

```

```
%Psi2 = fftshift(ifft(fftshift(Phi0)));

Psi2 = fft2(Phi0);


density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
%ticker = ticker + t/4;


A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;



P2 = abs(PsiOut.*conj(PsiOut));
%Con = sqrt(sum(P2).*dx);
%PsiOut = PsiOut/Con;
%P2 = abs(PsiOut.*conj(PsiOut));
p02 = abs(Psi0.*conj(Psi0));
imagesc(x,y,P2);
%plot(x,P2)
drawnow



end


%STAGE 3: Turn down Damping

dampticker = 0;
for(k =1:movemax2);

dampticker = dampticker + 1;

%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;

dampdown = .03 - ((.03-.003)/movemax2)*dampticker;

i2 = 1i-dampdown;

Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);

%ticker = ticker + t/4;
%Transform to momentum space

%Phi0 = fftshift(fft(fftshift(Psi1)));
Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
%ticker = ticker + t/2;

%Transform back to free space

%Psi2 = fftshift(ifft(fftshift(Phi0)));

Psi2 = fft2(Phi0);


density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm ).*t./i2/2);
%ticker = ticker + t/4;


```

```
A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;



P2 = abs(PsiOut.*conj(PsiOut));
%Con = sqrt(sum(P2).*dx);
%PsiOut = PsiOut/Con;
%P2 = abs(PsiOut.*conj(PsiOut));

subplot(2,1,1)
imagesc(x,y,P2);
subplot(2,1,2)
imagesc(x,y,angle(PsiOut));
%plot(x,P2)

p02 = abs(Psi0.*conj(Psi0));
%plot(x,P2)
%drawnow
end



%Generation of third Laser

ticker = 0;
for(k =1:movemax5);

    x3t = 0;
    y3t = 0;

Vmax = 1.2*E;
dtild = 2;

Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtild^2))*ticker/movemax5;

Vlasers = Vlaser3;

ticker = ticker + 1;

%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;


i2 = 1i-damp2;

Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlasers).*t./i2/2);
%ticker = ticker + t/4;
%Transform to momentum space

%Phi0 = fftshift(fft(fftshift(Psi1)));
Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
%ticker = ticker + t/2;

%Transform back to free space

%Psi2 = fftshift(ifft(fftshift(Phi0)));

Psi2 = fft2(Phi0);


density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
%ticker = ticker + t/4;


```

```
A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;



P2 = abs(PsiOut.*conj(PsiOut));
%Con = sqrt(sum(P2).*dx);
%PsiOut = PsiOut/Con;
%P2 = abs(PsiOut.*conj(PsiOut));
p02 = abs(Psi0.*conj(Psi0));
%imagesc(x,y,P2);

%plot(x,P2)
subplot(2,1,1)
imagesc(x,y,P2);
subplot(2,1,2)
imagesc(x,y,angle(PsiOut));
%plot(x,P2)

drawnow
end



%Stage 4: Moving the beams.

for(m = 1:4)
movemax4 = 800*(m==1)+834*(m==2)+(m==3)*915+(m==4)*950;


%STAGE 2: Generation of Laser

ticker = 0;
for(k =1:movemax3);

    x1t = -5;
    y1t = 0;
    x2t = -5;
    y2t = 0;
    x3t = 0;
    y3t = 0;

Vmax = 1.2*E;
dtild = .8;
dtildc = 2;
Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2))*ticker/movemax3;
Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2))*ticker/movemax3;
Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtildc^2));

ticker = ticker + 1;

Vlasers = Vlaser1 +Vlaser2 + Vlaser3;

%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;


i2 = 1i-damp1;

Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlasers).*t./i2/2);
%ticker = ticker + t/4;
%Transform to momentum space

%Phi0 = fftshift(fft(fftshift(Psi1)));
Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
%ticker = ticker + t/2;

%Transform back to free space
```

```
297
298 %Psi2 = fftshift(ifft(fftshift(Phi0)));
299
300 Psi2 = fft2(Phi0);
301
302
303 density = abs(Psi2.*conj(Psi2));
304 Interm = gmax *density;
305
306 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
307 %ticker = ticker + t/4;
308
309
310 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
311 PsiOut = PsiOut/A;
312
313
314
315 P2 = abs(PsiOut.*conj(PsiOut));
316 %Con = sqrt(sum(P2).*dx);
317 %PsiOut = PsiOut/Con;
318 %P2 = abs(PsiOut.*conj(PsiOut));
319
320 subplot(2,1,1);
321 imagesc(x,y,P2);
322 subplot(2,1,2);
323 imagesc(x,y,angle(PsiOut));
324
325 drawnow
326 end
327
328
329
330   %Beam movement
331 ticker = 0;
332 for(k =1:movemax4);
333
334
335
336      trot = ticker/movemax4;
337 %     w1 = pi;
338 %     w2 = pi/2;
339 %
340 %     r1 = -5 + 5*trot;
341 %     r2 = -5 -10*trot;
342 %
343
344
345     x3t = 0;
346     y3t = 0;
347
348     %P1
349
350
351     x1t = -5+5*trot;
352     y1t = -4*trot;
353
354     %P2
355
356     x2t = (-5+5*trot);
357     y2t = (4*trot);
358
359
360 Vmax = 1.2*E;
361 dtild = .8;
362
363
364 Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2));
365 Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2));
366 Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtildc^2));
367
```

```
Vlasers = Vlaser1 + Vlaser2 + Vlaser3;
ticker = ticker + 1;

%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;


i2 = 1i-damp2;
Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlasers).*t./i2/2);
%ticker = ticker + t/4;
%Transform to momentum space

%Phi0 = fftshift(fft(fftshift(Psi1)));
Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
%ticker = ticker + t/2;

%Transform back to free space

%Psi2 = fftshift(ifft(fftshift(Phi0)));

Psi2 = fft2(Phi0);


density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
%ticker = ticker + t/4;


A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;




P2 = abs(PsiOut.*conj(PsiOut));
%Con = sqrt(sum(P2).*dx);
%PsiOut = PsiOut/Con;
%P2 = abs(PsiOut.*conj(PsiOut));
p02 = abs(Psi0.*conj(Psi0));

subplot(2,1,1);
imagesc(x,y,P2);
subplot(2,1,2);
imagesc(x,y,angle(PsiOut));

%plot(x,P2)
drawnow
end

  %Beam movement
ticker = 0;
for(k =1:movemax8);



   trot = ticker/movemax8;


    x3t = 0;
    y3t = 0;

    %P1


    x1t = 0;
    y1t = 4-4*trot;
```

```
439
440     %P2
441
442     x2t = 10*trot;
443     y2t = -4;
444
445
446 Vmax = 1.2*E;
447 dtild = .8;
448 Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2));
449 Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2));
450 Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtildc^2));
451
452 Vlasers = Vlaser1 + Vlaser2 + Vlaser3;
453 ticker = ticker + 1;
454
455 %First Evolution step in free space
456
457 density = abs(PsiOut.*conj(PsiOut));
458 Interm = gmax * density;
459
460
461 i2 = 1i-damp2;
462 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlasers).*t./i2/2);
463 %ticker = ticker + t/4;
464 %Transform to momentum space
465
466 %Phi0 = fftshift(fft(fftshift(Psi1)));
467 Phi0=ifft2(Psi1);
468 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
469 %ticker = ticker + t/2;
470
471 %Transform back to free space
472
473 %Psi2 = fftshift(ifft(fftshift(Phi0)));
474
475 Psi2 = fft2(Phi0);
476
477
478 density = abs(Psi2.*conj(Psi2));
479 Interm = gmax *density;
480
481 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
482 %ticker = ticker + t/4;
483
484
485 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
486 PsiOut = PsiOut/A;
487
488
489
490
491 P2 = abs(PsiOut.*conj(PsiOut));
492 %Con = sqrt(sum(P2).*dx);
493 %PsiOut = PsiOut/Con;
494 %P2 = abs(PsiOut.*conj(PsiOut));
495 p02 = abs(Psi0.*conj(Psi0));
496
497 subplot(2,1,1);
498 imagesc(x,y,P2);
499 subplot(2,1,2);
500 imagesc(x,y,angle(PsiOut));
501
502 %plot(x,P2)
503 drawnow
504 end
505
506
507 %Beam Rampdown
508
509 ticker = 0;
```

```
510 for(k =1:movemax3);
511
512     x1t = 0;
513     y1t = 0;
514     x2t = 0;
515     y2t = 0;
516     x3t = 0;
517     y3t = 0;
518
519 Vmax = 1.2*E;
520 dtild = .8;
521 Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2))*(1-ticker/movemax3);
522 Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2))*(1-ticker/movemax3);
523 Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtildc^2));
524
525 ticker = ticker + 1;
526
527 Vlasers = Vlaser1 +Vlaser2 + Vlaser3;
528
529 %First Evolution step in free space
530
531 density = abs(PsiOut.*conj(PsiOut));
532 Interm = gmax * density;
533
534
535 i2 = 1i-damp1;
536
537 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlasers).*t./i2/2);
538 %ticker = ticker + t/4;
539 %Transform to momentum space
540
541 %Phi0 = fftshift(fft(fftshift(Psi1)));
542 Phi0=ifft2(Psi1);
543 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
544 %ticker = ticker + t/2;
545
546 %Transform back to free space
547
548 %Psi2 = fftshift(ifft(fftshift(Phi0)));
549
550 Psi2 = fft2(Phi0);
551
552
553 density = abs(Psi2.*conj(Psi2));
554 Interm = gmax *density;
555
556 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
557 %ticker = ticker + t/4;
558
559
560 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
561 PsiOut = PsiOut/A;
562
563
564
565 P2 = abs(PsiOut.*conj(PsiOut));
566 %Con = sqrt(sum(P2).*dx);
567 %PsiOut = PsiOut/Con;
568 %P2 = abs(PsiOut.*conj(PsiOut));
569
570 subplot(2,1,1);
571 imagesc(x,y,P2);
572 subplot(2,1,2);
573 imagesc(x,y,angle(PsiOut));
574
575 drawnow
576 end
577
578
579
580 end
```

```
581
582
583 % Generation of the pop laser
584
585 ticker = 0;
586 for(k =1:movemax5);
587
588     x3t = 0;
589     y3t = 0;
590
591 Vmax = 1.2*E;
592 Vmaxp = -2.4*E;
593 dtild = .8;
594 dtildc = 2;
595
596 Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtildc^2));
597
598 Vlaserp = Vmaxp*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtild^2))*ticker/movemax5;
599
600 Vlasers = Vlaser3 +Vlaserp;
601
602 ticker = ticker + 1;
603
604 %First Evolution step in free space
605
606 density = abs(PsiOut.*conj(PsiOut));
607 Interm = gmax * density;
608
609
610 i2 = 1i-damp2;
611
612 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlasers).*t./i2/2);
613 %ticker = ticker + t/4;
614 %Transform to momentum space
615
616 %Phi0 = fftshift(fft(fftshift(Psi1)));
617 Phi0=ifft2(Psi1);
618 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
619 %ticker = ticker + t/2;
620
621 %Transform back to free space
622
623 %Psi2 = fftshift(ifft(fftshift(Phi0)));
624
625 Psi2 = fft2(Phi0);
626
627
628 density = abs(Psi2.*conj(Psi2));
629 Interm = gmax *density;
630
631 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
632 %ticker = ticker + t/4;
633
634
635 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
636 PsiOut = PsiOut/A;
637
638
639
640 P2 = abs(PsiOut.*conj(PsiOut));
641 %Con = sqrt(sum(P2).*dx);
642 %PsiOut = PsiOut/Con;
643 %P2 = abs(PsiOut.*conj(PsiOut));
644 p02 = abs(Psi0.*conj(Psi0));
645 %imagesc(x,y,P2);
646
647 %plot(x,P2)
648 subplot(2,1,1)
649 imagesc(x,y,P2);
650 subplot(2,1,2)
651 imagesc(x,y,angle(PsiOut));
```

```
%plot(x,P2)

drawnow
end

ticker = 0;
for(k =1:movemax7);

    x3t = 0;
    y3t = 0;

Vmax = 1.2*E;
dtild = .8;
dtildc = 2;

Vlaser3 = Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtildc^2));

Vlaserp = -Vmax*exp(-2*((X-x3t).^2+(Y-y3t).^2)/(dtild^2));

Vlasers = Vlaser3 +Vlaserp;

ticker = ticker + 1;

%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;


i2 = 1i-damp2;

Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlasers).*t./i2/2);
%ticker = ticker + t/4;
%Transform to momentum space

%Phi0 = fftshift(fft(fftshift(Psi1)));
Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
%ticker = ticker + t/2;

%Transform back to free space

%Psi2 = fftshift(ifft(fftshift(Phi0)));

Psi2 = fft2(Phi0);


density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
%ticker = ticker + t/4;


A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;



P2 = abs(PsiOut.*conj(PsiOut));
%Con = sqrt(sum(P2).*dx);
%PsiOut = PsiOut/Con;
%P2 = abs(PsiOut.*conj(PsiOut));
p02 = abs(Psi0.*conj(Psi0));
%imagesc(x,y,P2);

%plot(x,P2)
subplot(2,1,1)
imagesc(x,y,P2);
subplot(2,1,2)
imagesc(x,y,angle(PsiOut));
```

```
%plot(x,P2)

drawnow
end
```

A.8 Code for 3.6

```
%function [x] = (BECInit,sigx,g2D);

% Initial Parameters

clear all


[BECInit,sigx,g2D] = TwoDBECGenCorrect();

N = length(sigx);
sigmax = 3.74E-6;
xmax = -2*min(sigx);
%xmax = 40/1.1;   %in scaled units
t = ( 2* xmax^2)/(pi^2*N^2)
damp1= 0.03;
damp2= 0.003;
%indexnum = [1:256];

%Movemax determines the number of loop iterations.
movemax1 = 1000;
movemax2 = 100;
movemax3 = 1000;
movemax4 = 3500;
movemax5 = 750;
movemax6 = 1000;
movemax7 = 8000;
movemax8 = 4000;


tmax = movemax1 * t;
gmax = g2D;
E = sqrt(g2D);

% X scaling.
dx = xmax/N;
nmid = floor (N/2);
v0 = [0:N-1];
x = (v0 * dx) - (xmax/2);
y = x;
[X,Y] = meshgrid(x,y);

%k Scaling.
kmax = (2 * pi)/(dx);
dk = kmax/N;
p = find(v0 > nmid);
vp = v0;
vp(p) = (N - v0(p));
eta = vp * dk;
nu = eta;

[Eta,Nu] = meshgrid(eta,nu);

%First transform from psi to phi
Psi0 = BECInit;

%Intial conitions before loop.
PsiOut = Psi0;
ticker = 0;

%STAGE 1: Stabilization of Ground state
```

```
61
62 for(k =1:movemax1);
63
64 %First Evolution step in free space
65
66 density = abs(PsiOut.*conj(PsiOut));
67 Interm = gmax * density;
68 i2 = 1i-damp1;
69 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
70
71 %Transform to momentum space
72
73 %Phi0 = fftshift(fft(fftshift(Psi1)));
74 Phi0=ifft2(Psi1);
75 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
76
77 %Transform back to free space
78
79 Psi2 = fft2(Phi0);
80 density = abs(Psi2.*conj(Psi2));
81 Interm = gmax *density;
82 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
83 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
84 PsiOut = PsiOut/A;
85 P2 = abs(PsiOut.*conj(PsiOut));
86 p02 = abs(Psi0.*conj(Psi0));
87
88
89 %Graphing Area
90
91 subplot(1,2,1)
92 xlabel('x Position [microns]');
93 ylabel('y Position [microns]');
94 title('Density')
95 imagesc(x,y,P2);
96 axis image
97 subplot(1,2,2)
98 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
99 imagesc(x,y,anglef);
100 xlabel('x Position [microns]');
101 ylabel('y Position [microns]');
102 title('Phase');
103 axis image
104 drawnow
105 end
106
107 %STAGE 3: Turn down Damping
108
109 dampticker = 0;
110 for(k =1:movemax2);
111
112 dampticker = dampticker + 1;
113
114 %First Evolution step in free space
115
116 density = abs(PsiOut.*conj(PsiOut));
117 Interm = gmax * density;
118 dampdown = .03 - ((.03-.003)/movemax2)*dampticker;
119 i2 = 1i-dampdown;
120 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
121
122 %Transform to momentum space
123
124 Phi0=ifft2(Psi1);
125 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
126
127 %Transform back to free space
128
129 Psi2 = fft2(Phi0);
130 density = abs(Psi2.*conj(Psi2));
131 Interm = gmax *density;
```

```
PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;
P2 = abs(PsiOut.*conj(PsiOut));


%Graphing Area

subplot(1,2,1)
xlabel('x Position [microns]');
ylabel('y Position [microns]');
title('Density')
imagesc(x,y,P2);
axis image
subplot(1,2,2)
anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
imagesc(x,y,anglef);
xlabel('x Position [microns]');
ylabel('y Position [microns]');
title('Phase');
axis image
drawnow


p02 = abs(Psi0.*conj(Psi0));

end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%LASER WORK%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Vholder = 0;

%Generation of the lasers
i = 1;
ticker = 0;
for(k =1:movemax5);

    x1t = -5;
    y1t = 0;

    x2t = -5;
    y2t = 0;

    Wsitex = 0;
    Wsitey = 8;
    Wdtild = 2;


    %%%%%%%%%%%%%%%%%%%% Pinning Lasers %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    xposit= (i ==1)*6+(i ==2)*3+(i ==3)*3+(i ==4)*3+(i ==5)*1+(i ==6)*1+(i ==7)*1+(i ==8)*(-1)+(
    yposit= (i ==1)*0+(i ==2)*3+(i ==3)*(-3)+(i ==4)*(0)+(i ==5)*(-5)+(i ==6)*(-2)+(i ==7)*(2)+(

   xt(i)= xposit;
   yt(i) =yposit;


Vmax = 3*E;
dtild = .8;


Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2))*ticker/movemax5;
Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2))*ticker/movemax5;
Vlaser(:,:,i) = Vmax*exp(-2*((X-xt(i)).^2+(Y-yt(i)).^2)/(dtild^2))*ticker/movemax5;
Vwaste = 2*Vmax*exp(-2*((X-Wsitex).^2+(Y-Wsitey).^2)/(Wdtild^2))*ticker/movemax5;

Vlasers = Vlaser1 + Vlaser2+ Vlaser(:,:,i)+Vholder+Vwaste;
ticker = ticker + 1;

%First Evolution step in free space
```

```
203
204 density = abs(PsiOut.*conj(PsiOut));
205 Interm = gmax * density;
206
207 i2 = 1i-damp2;
208 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
209
210 %Transform to momentum space
211
212 Phi0=ifft2(Psi1);
213 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
214
215 %Transform back to free space
216
217 Psi2 = fft2(Phi0);
218 density = abs(Psi2.*conj(Psi2));
219 Interm = gmax *density;
220 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
221 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
222 PsiOut = PsiOut/A;
223 P2 = abs(PsiOut.*conj(PsiOut));
224 p02 = abs(Psi0.*conj(Psi0));
225
226 %Graphing Area
227
228 subplot(1,2,1)
229 xlabel('x Position [microns]');
230 ylabel('y Position [microns]');
231 title('Density')
232 imagesc(x,y,P2);
233 axis image
234 subplot(1,2,2)
235 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
236 imagesc(x,y,anglef);
237 xlabel('x Position [microns]');
238 ylabel('y Position [microns]');
239 title('Phase');
240 axis image
241 drawnow
242 end
243
244 Vholder = Vlaser(:,:,i) + Vholder +Vwaste;
245
246 for(i=1:7)
247
248 %Beam Movement 1
249 ticker = 0;
250 for(k =1:movemax4);
251
252     trot = ticker/movemax4;
253
254     xposit= (i ==1)*6+(i ==2)*3+(i ==3)*3+(i ==4)*3+(i ==5)*1+(i ==6)*1+(i ==7)*1+(i ==8)*(-1);
255     yposit= (i ==1)*0+(i ==2)*3+(i ==3)*(-3)+(i ==4)*(0)+(i ==5)*(-5)+(i ==6)*(-2)+(i ==7)*(2)+(
256
257     x1t = -5+(5+xposit)*(trot).^(1/2);
258     y1t = 0+yposit*(trot).^(1/2);
259
260     %P2
261
262     x2t = -5+5*trot.^(5/10);
263     y2t = 7*trot.^(5/10);
264
265 Vmax = 3*E;
266 dtild = .8;
267 Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2));
268 Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2));
269
270 Vlasers = Vlaser1+Vlaser2+Vholder;
271 ticker = ticker + 1;
272
273 %First Evolution step in free space
```

```
274
275 density = abs(PsiOut.*conj(PsiOut));
276 Interm = gmax * density;
277
278 i2 = 1i-damp2;
279 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlasers).*t./i2/2);
280
281 %Transform to momentum space
282
283 Phi0=ifft2(Psi1);
284 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
285
286 %Transform back to free space
287
288 Psi2 = fft2(Phi0);
289 density = abs(Psi2.*conj(Psi2));
290 Interm = gmax *density;
291 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
292 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
293 PsiOut = PsiOut/A;
294 P2 = abs(PsiOut.*conj(PsiOut));
295 p02 = abs(Psi0.*conj(Psi0));
296
297
298 %Graphing Area
299
300 subplot(1,2,1)
301 xlabel('x Position [microns]');
302 ylabel('y Position [microns]');
303 title('Density')
304 imagesc(x,y,P2);
305 axis image
306 subplot(1,2,2)
307 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
308 imagesc(x,y,anglef);
309 xlabel('x Position [microns]');
310 ylabel('y Position [microns]');
311 title('Phase');
312 axis image
313 drawnow
314
315 end
316
317 %BEAM TRANSFER 1
318
319 ticker = 0;
320 for(k =1:movemax5);
321
322
323     xposit= (i ==1)*6+(i ==2)*3+(i ==3)*3+(i ==4)*3+(i ==5)*1+(i ==6)*1+(i ==7)*1+(i ==8)*(-1);
324     yposit= (i ==1)*0+(i ==2)*3+(i ==3)*(-3)+(i ==4)*(0)+(i ==5)*(-5)+(i ==6)*(-2)+(i ==7)*(2)+(
325
326     xpositu= (i ==1)*3+(i ==2)*3+(i ==3)*3+(i ==4)*1+(i ==5)*1+(i ==6)*1;
327     ypositu= (i ==1)*3+(i ==2)*(-3)+(i ==3)*(0)+(i ==4)*(-5)+(i ==5)*(-2)+(i ==6)*(2);
328
329     x1t = -5;
330     y1t = 0;
331     x2t = -5;
332     y2t = 0;
333
334     xt(i) = xposit;
335     yt(i)= yposit;
336
337     xt(i+1) = xpositu;
338     yt(i+1)= ypositu;
339
340
341
342 Vmax = 3*E;
343 dtild = .8;
344 downer = (1-ticker/movemax5);
```

```
345
346
347 %Laser Turn ons
348 Vlaser1 = Vmax*exp(-2*((X-x1t).^2+(Y-y1t).^2)/(dtild^2))*ticker/movemax5;
349 Vlaser2 = Vmax*exp(-2*((X-x2t).^2+(Y-y2t).^2)/(dtild^2))*ticker/movemax5;
350 Vlaser(:,:,i+1) = Vmax*exp(-2*((X-xt(i+1)).^2+(Y-yt(i+1)).^2)/(dtild^2))*ticker/movemax5;
351
352 %Laser Turn offs
353
354 Vholder1 =Vmax*exp(-2*((X-xt(i)).^2+(Y-yt(i)).^2)/(dtild^2))*downer;
355 Vlasers = Vlaser1 + Vlaser2+ Vlaser(:,:,i+1)+Vholder+Vholder1;
356 ticker = ticker + 1;
357
358 %First Evolution step in free space
359
360 density = abs(PsiOut.*conj(PsiOut));
361 Interm = gmax * density;
362 i2 = 1i-damp2;
363 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
364
365 %Transform to momentum space
366
367 Phi0=ifft2(Psi1);
368 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
369
370 %Transform back to free space
371
372 Psi2 = fft2(Phi0);
373 density = abs(Psi2.*conj(Psi2));
374 Interm = gmax *density;
375 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
376 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
377 PsiOut = PsiOut/A;
378 P2 = abs(PsiOut.*conj(PsiOut));
379 p02 = abs(Psi0.*conj(Psi0));
380
381 %Graphing Area
382
383 subplot(1,2,1)
384 xlabel('x Position [microns]');
385 ylabel('y Position [microns]');
386 title('Density')
387 imagesc(x,y,P2);
388 axis image
389 subplot(1,2,2)
390 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
391 imagesc(x,y,anglef);
392 xlabel('x Position [microns]');
393 ylabel('y Position [microns]');
394 title('Phase');
395 axis image
396 drawnow
397
398 end
399
400 Vholder = Vlaser(:,:,i+1)+Vholder;
401 movemax4 = (i ==1)*2500+(i ==2)*3000+(i ==3)*2500+(i ==4)*2250+(i ==5)*2000+(i ==6)*1500+(i ==7)
402 end
403 ticker = 0;
404 for(k =1:movemax8);
405
406
407 ticker = ticker + 1;
408
409 %First Evolution step in free space
410
411 density = abs(PsiOut.*conj(PsiOut));
412 Interm = gmax * density;
413 i2 = 1i-damp2;
414 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
415
```

```
%Transform to momentum space

Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);

%Transform back to free space

Psi2 = fft2(Phi0);
density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;
PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;
P2 = abs(PsiOut.*conj(PsiOut));
p02 = abs(Psi0.*conj(Psi0));

%Graphing Area

subplot(1,2,1)
xlabel('x Position [microns]');
ylabel('y Position [microns]');
title('Density')
imagesc(x,y,P2);
axis image
subplot(1,2,2)
anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
imagesc(x,y,anglef);
xlabel('x Position [microns]');
ylabel('y Position [microns]');
title('Phase');
axis image
drawnow

end

ticker = 0;
for(k =1:movemax7);


  Vholderd = Vholder*(1-ticker/movemax7);

Vmax = 3*E;
dtild = .8;
downer = (1-ticker/movemax5);

%Laser Turn ons

Vlasers =Vholderd;
ticker = ticker + 1;

%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;
i2 = 1i-damp2;
Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);

%Transform to momentum space


Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);

%Transform back to free space

Psi2 = fft2(Phi0);
density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;
PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;
```

```
P2 = abs(PsiOut.*conj(PsiOut));
p02 = abs(Psi0.*conj(Psi0));


%Graphing Area

subplot(1,2,1)
xlabel('x Position [microns]');
ylabel('y Position [microns]');
title('Density')
imagesc(x,y,P2);
axis image
subplot(1,2,2)
anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
imagesc(x,y,anglef);
xlabel('x Position [microns]');
ylabel('y Position [microns]');
title('Phase');
axis image
drawnow

end
```

A.9 Code for 3.7

```
%function [x] = (BECInit,sigx,g2D);

% Initial Parameters

clear all


[BECInit,sigx,g2D] = TwoDBECGenCorrect();

N = length(sigx);
sigmax = 3.74E-6;
xmax = -2*min(sigx);
%xmax = 40/1.1;   %in scaled units
t = 1*(2 * xmax^2)/(pi^2*N^2);
damp1= 0.03;
damp2= 0.003;
%indexnum = [1:256];

%Movemax determines the number of loop iterations.
movemax1 = 1000;
movemax2 = 100;
movemax3 = 1000;
movemax4 = 2000;
movemax5 = 750;
movemax6 = 1000;
movemax7 = 800;


tmax = movemax1 * t;
gmax = g2D;
E = sqrt(g2D);

% X scaling.
dx = xmax/N;
nmid = floor (N/2);
v0 = [0:N-1];
x = (v0 * dx) - (xmax/2);
y = x;
[X,Y] = meshgrid(x,y);

%k Scaling.
kmax = (2 * pi)/(dx);
```

```
dk = kmax/N;
p = find(v0 > nmid);
vp = v0;
vp(p) = (N - v0(p));
eta = vp * dk;
nu = eta;

[Eta,Nu] = meshgrid(eta,nu);

%First transform from psi to phi

Psi0 = BECInit;

%Intial conitions before loop.
PsiOut = Psi0;
ticker = 0;
%density1 = abs(PsiOut.*conj(PsiOut));

anglepos(1) = 0;
numcirc = 6;
radial = 6;

angleplus = 2*pi/numcirc;

xcircpos(1) = radial*cos(anglepos(1));
ycircpos(1) = radial*sin(anglepos(1));


for(i =2:numcirc);

anglepos(i) = anglepos(i-1) + angleplus;

xcircpos(i) = radial*cos(anglepos(i));
ycircpos(i) = radial*sin(anglepos(i));

end




%%%%%%%%%%%%%%%%%%STAGE 1: Stabilization of Ground state%%%%%%%%%%%%%%%%%%

for(k =1:movemax1);


%First Evolution step in free space

density = abs(PsiOut.*conj(PsiOut));
Interm = gmax * density;
i2 = 1i-damp1;
Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);

%Transform to momentum space

Phi0=ifft2(Psi1);
Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);

%Transform back to free space

Psi2 = fft2(Phi0);

density = abs(Psi2.*conj(Psi2));
Interm = gmax *density;

PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
PsiOut = PsiOut/A;
P2 = abs(PsiOut.*conj(PsiOut));

%Graphing Area
```

```
114
115 subplot(1,2,1)
116 xlabel('x Position [microns]');
117 ylabel('y Position [microns]');
118 title('Density')
119 imagesc(x,y,P2);
120 axis image
121 subplot(1,2,2)
122 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
123 imagesc(x,y,anglef);
124 xlabel('x Position [microns]');
125 ylabel('y Position [microns]');
126 title('Phase');
127 axis image
128 drawnow
129
130
131
132
133 end
134
135 %STAGE 3: Turn down Damping
136
137 dampticker = 0;
138 for(k =1:movemax2);
139
140 dampticker = dampticker + 1;
141
142 %First Evolution step in free space
143
144 density = abs(PsiOut.*conj(PsiOut));
145 Interm = gmax * density;
146 dampdown = .03 - ((.03-.003)/movemax2)*dampticker;
147 i2 = 1i-dampdown;
148 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
149
150 %Transform to momentum space
151
152 Phi0=ifft2(Psi1);
153 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
154
155 %Transform back to free space
156
157 Psi2 = fft2(Phi0);
158 density = abs(Psi2.*conj(Psi2));
159 Interm = gmax *density;
160 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm).*t./i2/2);
161 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
162 PsiOut = PsiOut/A;
163 P2 = abs(PsiOut.*conj(PsiOut));
164
165 %Graphing Area
166
167 subplot(1,2,1)
168 xlabel('x Position [microns]');
169 ylabel('y Position [microns]');
170 title('Density')
171 imagesc(x,y,P2);
172 axis image
173 subplot(1,2,2)
174 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
175 imagesc(x,y,anglef);
176 xlabel('x Position [microns]');
177 ylabel('y Position [microns]');
178 title('Phase');
179 axis image
180 drawnow
181
182 end
183
184
```

```
185
186 %%%%%%%%%%%%%%%%%%%%%%%%%%%%LASER WORK%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
187
188
189 %Generation of the lasers
190 Vlasers = 0;
191 ticker = 0;
192 Vlasersh = 0;
193
194 Vmax = 1.2*E;
195 dtild = .8;
196 dtildc = 1.5;
197
198 %Generates initial positions
199 for(p =1:numcirc);
200
201 Vlaser(:,:,p) = Vmax*exp(-2*((X-xcircpos(p)).^2+(Y-ycircpos(p)).^2)/(dtild^2));
202 Vlaser2(:,:,p) = Vmax*exp(-2*((X-xcircpos(p)).^2+(Y-ycircpos(p)).^2)/(dtild^2));
203 Vlasersh = Vlasersh + Vlaser(:,:,p)+Vlaser2(:,:,p);
204 end
205
206 Vlasercent = Vmax*exp(-2*((X-0).^2+(Y-0).^2)/(dtildc^2));
207 Vlasersh = Vlasersh + Vlasercent;
208
209 %Turning on beams
210 for(k =1:movemax5);
211 Vlasers = Vlasersh*ticker/movemax5;
212 ticker = ticker + 1;
213
214 %First Evolution step in free space
215
216 density = abs(PsiOut.*conj(PsiOut));
217 Interm = gmax * density;
218 i2 = 1i-damp2;
219 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
220
221 %Transform to momentum space
222
223 Phi0=ifft2(Psi1);
224 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
225
226 %Transform back to free space
227
228 Psi2 = fft2(Phi0);
229 density = abs(Psi2.*conj(Psi2));
230 Interm = gmax *density;
231 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
232 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
233 PsiOut = PsiOut/A;
234
235 P2 = abs(PsiOut.*conj(PsiOut));
236 p02 = abs(Psi0.*conj(Psi0));
237
238 %Graphing Area
239
240 subplot(1,2,1)
241 xlabel('x Position [microns]');
242 ylabel('y Position [microns]');
243 title('Density')
244 imagesc(x,y,P2);
245 axis image
246 subplot(1,2,2)
247 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
248 imagesc(x,y,anglef);
249 xlabel('x Position [microns]');
250 ylabel('y Position [microns]');
251 title('Phase');
252 axis image
253 drawnow
254
255 end
```

```
256
257     %Beam Movement 1
258
259 ticker = 0;
260 Vlasers = 0;
261
262
263 for(k =1:movemax4);
264
265    rotinc = pi/movemax4;
266    radec = radial/movemax4;
267
268     trot = ticker/movemax4;
269     downer = 1-trot
270
271
272 Vmax = 1.2*E;
273 dtild = .8;
274 dtildc = 1.5;
275
276 %Creates time updated positions
277
278
279 for(i =1:numcirc);
280
281     anglepos(i) = anglepos(i) +rotinc;
282     xcircpos(i) = radial*cos(anglepos(i));
283     ycircpos(i) = radial*sin(anglepos(i));
284     Vlaser(:,:,i) = Vmax*exp(-2*((X-xcircpos(i)).^2+(Y-ycircpos(i)).^2)/(dtild^2));
285
286     xcircpos2(i) = (downer*radial)*cos(anglepos(i));
287     ycircpos2(i) = (downer*radial)*sin(anglepos(i));
288     Vlaser2(:,:,i) = Vmax*exp(-2*((X-xcircpos2(i)).^2+(Y-ycircpos2(i)).^2)/(dtild^2));
289
290 end
291 Vlasercent = Vmax*exp(-2*((X-0).^2+(Y-0).^2)/(dtildc^2));
292 Vlasers = sum(Vlaser,3)+sum(Vlaser2,3)+Vlasercent;
293
294 ticker = ticker + 1;
295
296 %First Evolution step in free space
297
298 density = abs(PsiOut.*conj(PsiOut));
299 Interm = gmax * density;
300 i2 = 1i-damp2;
301 Psi1 = PsiOut.*exp((.5*(X.^2+Y.^2)+Interm + Vlasers).*t./i2/2);
302
303 %Transform to momentum space
304
305 Phi0=ifft2(Psi1);
306 Phi0 = Phi0.*exp(.5.*(Nu.^2+Eta.^2).*t/i2);
307
308 %Transform back to free space
309
310 Psi2 = fft2(Phi0);
311 density = abs(Psi2.*conj(Psi2));
312 Interm = gmax *density;
313 PsiOut = Psi2.*exp((.5*(X.^2+Y.^2)+Interm+Vlasers).*t./i2/2);
314 A = sqrt(sum(sum(abs(PsiOut).^2)))*dx;
315 PsiOut = PsiOut/A;
316 P2 = abs(PsiOut.*conj(PsiOut));
317 p02 = abs(Psi0.*conj(Psi0));
318
319 %Graphing Area
320
321 subplot(1,2,1)
322 xlabel('x Position [microns]');
323 ylabel('y Position [microns]');
324 title('Density')
325 imagesc(x,y,P2);
326 axis image
```

```
327 subplot(1,2,2)
328 anglef = (angle(PsiOut)).*((max(max(density))*.01)<=P2);
329 imagesc(x,y,anglef);
330 xlabel('x Position [microns]');
331 ylabel('y Position [microns]');
332 title('Phase');
333 axis image
334 drawnow
335
336 end
```

Bibliography

[1] E. C. C. Samson. *Generating and manipulating quantized vortices in highly oblate Bose-Einstein condensates*. PhD thesis, The University of Arizona, 2012.

[2] *Bose-Einstein Condensation in Dilute Gases*. Cambridge University Press, 40 West 20th Street, New York, NY 10011-4211, USA, 2002.

[3] *Atom Optics*. Springer, 40 West 20th Street, New York, NY 10011-4211, USA, 2001.

[4] Weizhu Bao, Dieter Jaksch, and Peter A. Markowich. Numerical solution of the grosspitaevskii equation for boseeinstein condensation. *Journal of Computational Physics*, 187(1):318 – 342, 2003. ISSN 0021-9991. doi: http://dx.doi.org/10.1016/S0021-9991(03)00102-5. URL `http://www.sciencedirect.com/science/article/pii/S0021999103001025`.

[5] 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*, 62:1382–1389, Jul 2000. doi: 10.1103/PhysRevE.62.1382. URL `http://link.aps.org/doi/10.1103/PhysRevE.62.1382`.

[6] J. Javanainen and J. Ruostekoski. LETTER TO THE EDITOR: Symbolic calculation in development of algorithms: split-step methods for the Gross Pitaevskii equation. *Journal of Physics A Mathematical General*, 39:L179–L184, March 2006. doi: 10.1088/0305-4470/39/12/L02.

[7] Weizhu Bao and Hanquan Wang. An efficient and spectrally accurate numerical method for computing dynamics of rotating boseeinstein condensates. *Journal of Computational Physics*, 217(2):612 – 626, 2006. ISSN 0021-9991. doi: http://dx.doi.org/10.1016/j.jcp.2006.01.020. URL `http://www.sciencedirect.com/science/article/pii/S0021999106000258`.

[8] Mechthild Thalhammer, Marco Caliari, and Christof Neuhauser. High-order time-splitting hermite and fourier spectral methods. *Journal of Computational Physics*, 228(3):822 – 832, 2009. ISSN 0021-9991. doi: http://dx.doi.org/10.1016/j.jcp.2008.10.008. URL `http://www.sciencedirect.com/science/article/pii/S0021999108005251`.

[9] Erich Zauderer. Complex argument hermite-gaussian and laguerre-gaussian beams. *J. Opt. Soc. Am. A*, 3(4):465–469, Apr 1986. doi: 10.1364/JOSAA.3.000465. URL `http://josaa.osa.org/abstract.cfm?URI=josaa-3-4-465`.

[10] Uwe R. Fischer and Gordon Baym. Vortex states of rapidly rotating dilute bose-einstein condensates. *Phys. Rev. Lett.*, 90:140402, Apr 2003. doi: 10.1103/PhysRevLett.90.140402. URL `http://link.aps.org/doi/10.1103/PhysRevLett.90.140402`.