GROSS-PITAEVSKII SYSTEM: BOSE-EINSTEIN CONDENSATE

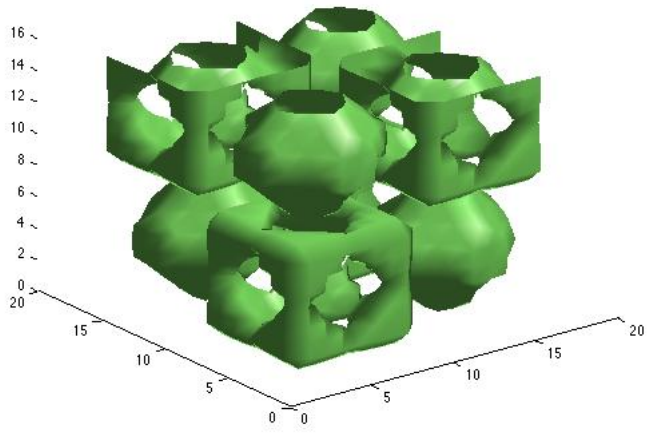
Andrew K. Walker

11 December, 2013

Abstract

An investigation and numerical approach to solving the Gross-Pitaevskii equations to model the Bose-Einstein Condensation problem. We explore the theoretical background of the equation and the physical system. Then we approach a numerical solution to the partial differential equations which model the system by using a spectral method, the discrete Fourier transform, to efficiently compute and visualize a solution.

Contents



1 Introduction

Partial differential equations are solved much quicker and to a greater level of accuracy using spectral methods than when using direct methods of solution. They are often limited by boundary conditions, but when appropriate these spectral methods drastically speed up the computation of a solution. In the case of the Gross-Pitaevskii system we will investigate a system with periodic boundary conditions, perfect for the Fast Fourier Transform. This system models a Bose-Eiinstein condensate in three dimensions.

2 Theoretical Background

2.1 The Bose-Einstein Condensation Problem

The Gross-Pitaevskii system, also known as the a nonlinear Schrödingier equation with potential, models a condensed state of matter, known as Bose-Eienstein condensate. The particular equation we will be studying is,

$$i\psi_t + \frac{1}{2}\nabla^2\psi - |\psi|^2\psi + [A_1\sin^2(x) + B_1][A_2\sin^2(y) + B_2][A_3\sin^2(z) + B_3]\psi = 0 \tag{1}$$

where $\nabla^2 = \partial_x^2 + \partial_y^2 + \partial_z^2$. Bose-Einstein condensate is a state of matter of dilute gas of bosons cooled to temperatures very close to absolute zero. A large fraction of the bosons occupy the lowest quantum state, in this state the quantum affects become apparent on a macroscopic scale. The first gaseous condensate was produced by Eric Cornell and Carl Wieman at the University of Colorado at Boulder NIST-JILA lab. They used a gas made of rubidium atoms cooled to 170 nanokelvin. This achievement earned Cornell and Wieman the Nobel Prize in Physics in 2001. The Gross-Piataevskii equation specifically uses the Hartree-Fock approximation and the pseudo potential interaction model. Here ψ is the wave function of the bosons, in a Bose-Einstein condensate the group of bosons all have the same quantum state and can be described by the same wave equation, this is the reason for the macroscopic quantum affects. The $|\psi|^2$ term is real and corresponds to the probability density of finding a particle in a given place at a given time.

The dynamics of the N identical pairwise interacting quantum particles is governed by the time-dependent N-body Schrdingier equation,

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar^2}{2m}\Delta^N\psi + \sum_{i,h=1,i\neq j}^{N} W(\mathbf{x}_i - \mathbf{x}_j)\psi + \sum_{i=1}^{N} V(\mathbf{x}_i)\psi \tag{2}$$

Then using the Lagrangian reduction technique and the Hartree-Fock approximation we can derive the Gross-Pitaevskii system above.

2.2 Numerical Algorithm Development & Implementation

One method for solving partial differential equations is to transform them into a spectral domain through a Fourier transformation. The one-dimensional Fourier transformation is defined, for $x \in (-\infty, \infty)$, by,

$$\widehat{f}(k) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} f(x)e^{-ikx}\,\mathrm{dx},$$

$$f(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} \widehat{f}(k)e^{ikx}\,\mathrm{dk}.$$

This is useful for computing solutions to partial differential equations because of the way derivatives are transformed in the Fourier domain, they follow the following property,

$$\widehat{f^{(n)}}(k) = (ik)^n \widehat{f}(k), \tag{3}$$

where $\widehat{f^{(n)}}$ is the nth derivative of f transformed into the Fourier domain. We will now present a proof of (3).

Proof. This will be a proof by induction. We will first show the base case, $n = 1$.

$$\begin{aligned}
\widehat{f^{(1)}}(k) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f^{(1)}(x) e^{-ikx} \,\mathrm{dx} \\
&= \frac{1}{\sqrt{2\pi}} \left[e^{-ikx} f(x) \Big|_{-\infty}^{\infty} + ik \int_{-\infty}^{\infty} f(x) e^{-ikx} \,\mathrm{dx} \right] \\
&= 0 + \frac{ik}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-ikx} \,\mathrm{dx} \\
\widehat{f^{(1)}}(k) &= (ik) \widehat{f}(k).
\end{aligned}$$

Now assume that this derivative property holds for the $(n-1)$th derivative of f, that is, $\widehat{f^{(n-1)}}(k) = (ik)^{n-1} \widehat{f}(k)$. Then it follows that,

$$\begin{aligned}
\widehat{f^{(n)}}(k) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f^{(n)}(x) e^{-ikx} \,\mathrm{dx} \\
&= \frac{1}{\sqrt{2\pi}} \left[e^{-ikx} f^{(n-1)}(x) \Big|_{-\infty}^{\infty} + ik \int_{-\infty}^{\infty} f^{(n-1)}(x) e^{-ikx} \,\mathrm{dx} \right] \\
&= 0 + (ik)(ik)^{n-1} \widehat{f}(k) \\
\widehat{f^{(n)}}(k) &= (ik)^n \widehat{f}(k).
\end{aligned}$$

Therefore, by the inductive hypothesis, the Fourier derivative property works for all $n \in \mathbb{N}$. □

In a numerical scheme, this Fourier transform must be applied to a finite domain of discrete points with periodic boundaries. This where the discrete Fourier transform comes in. The discrete Fourier transform takes the nth component of an $N \times 1$ vector, $\mathbf{x}$, and sends it to the kth component of the vector, $\mathbf{w}$, as follows,

$$w_k = \sum_{n=1}^{N} x_n e^{-2\pi i \frac{(k-1)(n-1)}{N}}, \text{ where } k \in \{1, 2, \ldots, N\}.$$

This transform can be performed with a single matrix operation, $\mathbf{w} = F_N \mathbf{x}$, where the (jk)th element of F_N is given by,

$$(F_N)_{jk} = f_n^{jk} = e^{2\pi i \frac{jk}{n}}.$$

The Fast Fourier Transform, or FFT, can do this in a minimal number of operations by noticing that,

$$(f_n^{jk})^2 = (e^{2\pi i \frac{jk}{2n}})^2 = e^{4\pi i \frac{jk}{2n}} = e^{2\pi i \frac{jk}{n}} = f_n^{jk}.$$

This will let us split F_N into two parts, an even and an odd component.

$$\mathbf{w}^{\text{even}} = F_{N/2} \mathbf{x}^{\text{even}} \text{ and}$$

$$\mathbf{w}^{\text{odd}} = F_{N/2}\mathbf{x}^{\text{odd}}$$

where,

$$\mathbf{x}^{\text{even}} = \begin{pmatrix} x_0 \\ x_2 \\ x_4 \\ \vdots \\ x_{N-2} \end{pmatrix}, \quad \mathbf{x}^{\text{odd}} = \begin{pmatrix} x_1 \\ x_3 \\ x_5 \\ \vdots \\ x_{N-1} \end{pmatrix}.$$

When N is a power of 2, we can continue to split the vectors in the same manner until we are left with an $\mathbf{x}^{\text{even}}$ and $\mathbf{x}^{\text{odd}}$ such that both are 1×1. Then every element of $\mathbf{w}$ can be reconstructed by,

$$w_k = w_k^{\text{even}} + f_N^k w_k^{\text{odd}}.$$

Implementing the discrete Fourier transformation using this method is highly efficient, it is an $\mathcal{O}(N \log N)$.

We will implement the Gross-Pitaevskii system (1) using the MATLAB command "fftn", which is the multi-dimensional FFT. This method is $\mathcal{O}(N \log N)$ and will map the domain $x \in [-\frac{L}{2}, \frac{L}{2}]$ to $x \in [\frac{-L}{2}, 0] \cup [0, \frac{L}{2}]$. The inverse transform, "ifftn", will transform back from the spectral domain.

We can rearrange (1) to get the equation,

$$\psi_t = i(\frac{1}{2}\nabla^2\psi - |\psi|^2\psi + [A_1 \sin^2(x) + B_1][A_2 \sin^2(y) + B_2][A_3 \sin^2(z) + B_3]\psi) \tag{4}$$

This is the function we will work with in our numerical methods. We notice that there are linear and non-linear terms, so we have to compute some of the terms before transforming them into Fourier space, while for others we can do our computations in Fourier space. The Fourier transform of (4) will look like,

$$\mathcal{F}(\psi_t) = i\left(-\frac{1}{2}K * \mathcal{F}(\psi) - \mathcal{F}\left(|\psi|^2\psi\right) + \mathcal{F}\left([A_1 \sin^2(x) + B_1][A_2 \sin^2(y) + B_2][A_3 \sin^2(z) + B_3]\psi\right)\right), \tag{5}$$

where K is the wavenumber vector and $\mathcal{F}$ is the Discrete Fourier Transform. In one dimension K would look like,

$$K = \frac{2\pi}{L}\left[0 : \frac{N}{2} - 1,\ -\frac{N}{2} : -1\right],$$

where L is the length of the domain and N is the number of grid points. In three dimensions we simply generalize this vector along the x, y, and z directions to end up with our K. Once we have this set up, we simply time step with MATLAB's "ode45".

3 Computational Results

For the computation the initial conditions were, $x, y, z \in [-\pi, \pi]$, $n = 16$, tspan$= 0 : .5 : 4$, $A_i = -1$, $B_i = -A_i$, and $\psi(x, y, z) = \cos(x)\cos(y)\cos(z)$. Also, a computation was run for $\psi(x, y, z) = \sin(x)\sin(y)\sin(z)$.

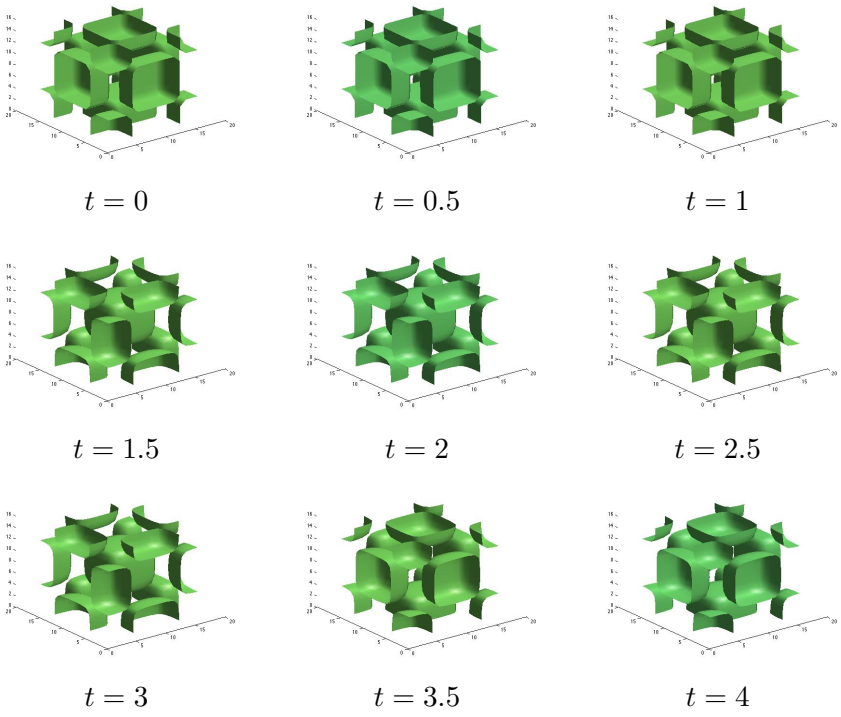


Figure 1: Gross-Pitaevskii with initial condition $\psi(x, y, z) = \cos(x)\cos(y)\cos(z)$.

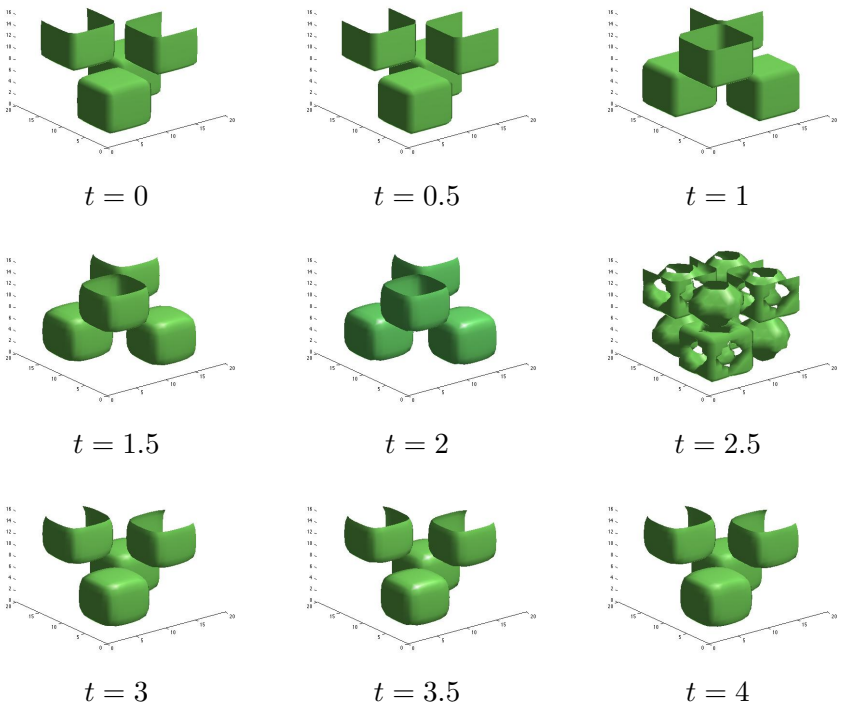


Figure 2: Gross-Pitaevskii with initial condition $\psi(x, y, z) = \sin(x)\sin(y)\sin(z)$.

4 Appendix A - MATLAB Code

```
%% Preamble

% Andrew Walker
% AMATH 481A - Autumn 2013
% Final Project - Bose-Einstein Condensate

close all;
clear all;

%% Initialize Variables

tspan = 0:.5:4;
n = 16;
L = 2*pi;
x = linspace(-L/2, L/2, n+1);
xspan = x(1:n);
yspan = xspan;
zspan = yspan;

[X,Y,Z] = meshgrid(xspan,yspan,zspan);

A1 = -1;
A2 = -1;
A3 = -1;

B1 = -A1;
B2 = -A2;
B3 = -A3;

% Create the K matrix

kx = (2*pi/L)*[0:(n/2-1) (-n/2):-1];
ky = kx;
kz = ky;

[KX,KY,KZ] = meshgrid(kx,ky,kz);

K = KX.^2+KY.^2+KZ.^2;

% Initial Condition psi=cos(x)cos(y)cos(z)
psi1_init = cos(X).*cos(Y).*cos(Z);

% Initial Condition psi=sin(x)sin(y)sin(z)
psi2_init = sin(X).*sin(Y).*sin(Z);

psi1_init_f = fftn(psi1_init);
psi2_init_f = fftn(psi2_init);

psi1_init_vec_f = reshape(psi1_init_f,n^3,1);
psi2_init_vec_f = reshape(psi2_init_f,n^3,1);
```

```
%% Use ODE45 to time-step

[t1,psi1_sol_vec_f] = ode45(@(t,psi_vec_f) rhsfft(t,psi_vec_f,K,A1,...
                                          A2,A3,B1,B2,B3,X,Y,Z,n),...
                            tspan,psi1_init_vec_f);

[t2,psi2_sol_vec_f] = ode45(@(t,psi_vec_f) rhsfft(t,psi_vec_f,K,A1,...
                                          A2,A3,B1,B2,B3,X,Y,Z,n),...
                            tspan,psi2_init_vec_f);
```

```
function rhs_vec = rhsfft(t,psi_vec_f,K,A1,A2,A3,B1,B2,B3,X,Y,Z,n)

    psi_f = reshape(psi_vec_f,n,n,n);
    psi = ifftn(psi_f);

    rhs = 1i*(.5*-K.*psi_f - ...
              fftn((abs(psi).^2).*psi) + ...
              fftn((A1*((sin(X)).^2)+B1).*...
                   (A2*((sin(Y)).^2)+B2).*...
                   (A3*((sin(Z)).^2)+B3).*psi));

    rhs_vec = reshape(rhs,n^3,1);

end
```