Jonathan McCormack
12/10/2025
AMATH 581

Evolution of a three-dimensional Bose Einstein Condensate on a Periodic Lattice

Background

In 1924, Satyendra Nath Bose identified that the Maxwell-Boltzmann distribution inadequately described the behavior of photons (then light quanta). Along with Albert Einstein, he published a paper that introduced the field of Bose-Einstein Statistics. Einstein generalized this to matter and theorized that some subatomic particles, when cooled, can occupy the same quantum state, condensing into a new form of matter. This new form of matter is the Bose-Einstein Condensate (BEC).

The Nonlinear Schrodinger equation describes the wave function (or probability density) of particles existing in a particular quantum state in a particular space and time. The Gross – Pitaevskii equation (GPE), named for Eugene P. Gross and Lev Petrovich Pitaevskii, is a simplified form of the Nonlinear Schrodinger equation that describes the wavefunction of a BEC where temperatures are so cold almost all particles have condensed to the ground state.

Gross – Pitaevskii equation:

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

The GPE solved for Ψ_t

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

Solution Approach

Pseudocode:

```
AMATH Final Project Pseudocode
-Setup meshgrid for three dimensional spatial domain
-Setup meshgrid for three dimensional spectral domain
-Setup right-hand side(rhs) function for the Gross-Pitaevskii system
-Set initial values for the wavefunction, ψ(x, y, z), evaluated at all points in the spatial domain
-use three dimension Fast Fourier Transform on initial values
-reshape initial values into a one dimension vector

-Evaluate IVP
    -reshape initial vector into three dimension array
    -calculate the rhs function for the Gross-Pitaevskii at the current time
    -reshape rhs vector to one dimension vector
    -time step until complete
```

The provided problem statement asked to solve the Partial Differential Equations (PDE) assuming periodic boundary conditions. Because the boundary conditions are Periodic instead of Neumann or Dirichlet, the PDE may be solved using the Fast Fourier Transform (FFT), which is not only more computationally efficient, but also spectrally accurate (more accurate than finite difference methods). Before the spectral method can be used, the initial condition of $\Psi(x,y,z)$ is calculated, processed through the three-dimensional FFT (using numpy.fft.fftn), and then reshaped to a one-dimensional vector for use in the scipy.integrate.solve_ivp function. Inside the solve_ivp function, the Ψ vector will be reshaped to a three-dimensional array and used to solve the GPE to determine Ψ_t for each position in the domain; the three-dimensional Ψ_t array will then be reshaped back to a one-dimensional array for use in iterating solve_ivp.

Jonathan McCormack
12/10/2025
AMATH 581

Because the GPE includes linear and nonlinear components, during the solve_ivp loop, the three-dimensional Ψ array has to be inverted to the spatial domain (using numpy.fft.ifftn) to calculate the nonlinear component which is then reverted back to the spectral domain (again using numpy.fft.fftn). The nonlinear component involves multiplying the probability density ($|\Psi|^2$) minus the Lattice array by the Ψ array. The Lattice array is the standing wave external to the initial conditions (for this simulation, it is unaffected by time). The linear component, the Laplacian, is calculated by multiplying the K array with the Ψ array.

Python Code for defining the PDE:

```python
# Define Spectral Method functions
# Set Lattice Matix (unique to Gross-Pitaevskii system)
Lattice = (A[0]*np.sin(X)**2 + B[0]) * (A[1]*np.sin(Y)**2 + B[1]) * (A[2]*np.sin(Z)**2 + B[2])

# Define FFT equation
def fftrhs(t, psihatvec, K, Lattice):

    psihat = psihatvec.reshape((n, n, n), order = 'F')
    psi = np.fft.ifftn(psihat)
    nonlinearhat = np.fft.fftn((np.abs(psi)**2)*psi - Lattice*psi)

    rhs = (-1j*(0.5*K*psihat + nonlinearhat)).reshape(n**3, order = 'F')
    return rhs
```

Implementation

The below code snippet was the section of the preamble that established the initial conditions. The code was written so that it would work regardless of the users changes to L (the Length of the spatial domain), n (the number of internal points), tspan (the start and en time), and the attraction/repulsion constants (A and B). The code implemented the numpy FFT package for the implementation of spectral methods. The spectral domain was defined using np.fft.fftfreq, rather than concatenating explicitly defined spectral points as in class. The biggest efficiency savings were gained by calculating the K mesh grid below and the Lattice array outside of the solve_ivp loop. The GPE was evaluated for the following initial conditions: Part A: ψ(x, y, z) = cos(x)*cos(y)*cos(z) and Part B: ψ(x, y, z) = sin(x) sin(y) sin(z).

Python Code for setting Initial Conditions:

```python
# Set the initial conditions
L = 2*np.pi # Half Length of Spatial Domain
n = 16 # Number of internal points
tspan = [0, 4]; dt = 0.5 # Set time endpoints and time step
A = np.array([-1, -1, -1])
B = -A

# Set Time Grid
t_eval = np.arange(tspan[0], tspan[1] + dt, dt)

# Set Spatial Mesh Grid
x = np.linspace(-L/2, L/2, n+1)[:n]
[X, Y, Z] = np.meshgrid(x, x, x)

# Set Spectral Mesh Grid
k = 2*np.pi*np.fft.fftfreq(n, d=L/n)
k[0] = 1e-6
[KX, KY, KZ] = np.meshgrid(k, k, k)
K = KX**2 + KY**2 + KZ**2
```

Jonathan McCormack
12/10/2025
AMATH 581

Visualization

Method

The squared absolute value of the wave function shows the probability density of finding a particle at any particular location. In order to find this probability density, the complex solution from solve_ivp was reshaped to separate three-dimensional arrays for each time point and inverted by np.fft.ifftn. The absolute value of the resultant array was squared element-wise. These nine 16x16x16 arrays represent the three-dimensional probability density for each solution at each time point.

Ideally, the GPE is plotted in such a way as to show where the probability density is at or just below 1.0. Matplotlib lacks the sufficient three-dimensional rendering capability for probability density (Isosurface) plots. Alternatively, one could use the Marching Cube algorithm, but the plot would only show locations with a probability density of exactly 1.0. In order to handle the lack of three-dimensional capability, the dimension of the solution could be reduced and graphed with a Matplotlib slice or contour plot, but this would not have shown the evolution in time and space.

Alternatively, the below plots were generated using Plotly Graphic Objects to show the gradient in probability density. The Isosurface shows all locations with a probability density greater than 0.25, clearly showing the regions of highest density. The opacity was set to 0.5 so the user could view the entire gradient. The surface count was set to 20 to increase resolution without affecting rendering speed. The solutions were iterated across all time points to show the clear change in time and space. The resultant images were saved as PNGs (Plotly does not cross-communicate with Matplotlib for printed subplots as a single image). Lastly, the below plots used the default 'Plasma' color map to improve visual coolness.

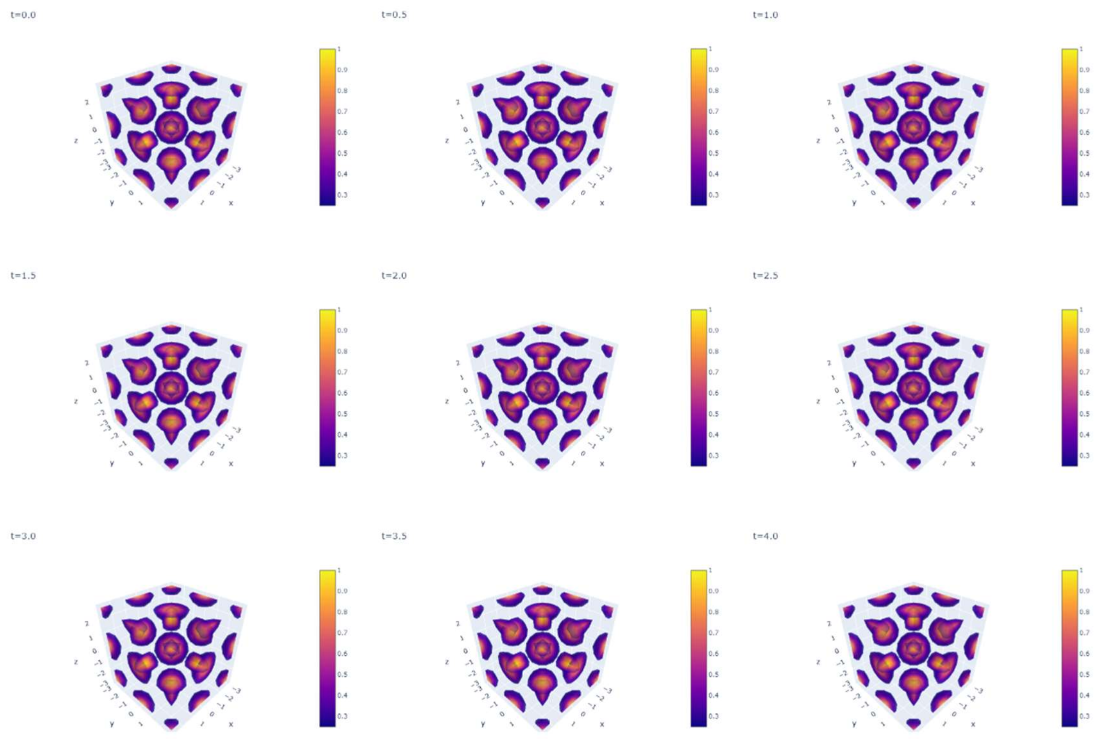
Python Code for Isosurfaces using Plotly:

```python
# Plot Isosurfaces for Part A: Initial Condition:  ψ(x, y, z) = cos(x)*cos(y)*cos(z)
for i in range(len(t_eval)):
    image = (np.abs(np.fft.ifftn(sol1.y.T[i,:].reshape(n,n,n, order = 'F'))))**2
    fig = go.Figure(data=go.Isosurface(x=X.flatten(), y=Y.flatten(), z=Z.flatten(), value=image.flatten(),
                                       opacity=0.5, isomin=0.25, isomax=1, surface_count=20))
    fig.write_image(f"cos{t_eval[i]}.png")
```

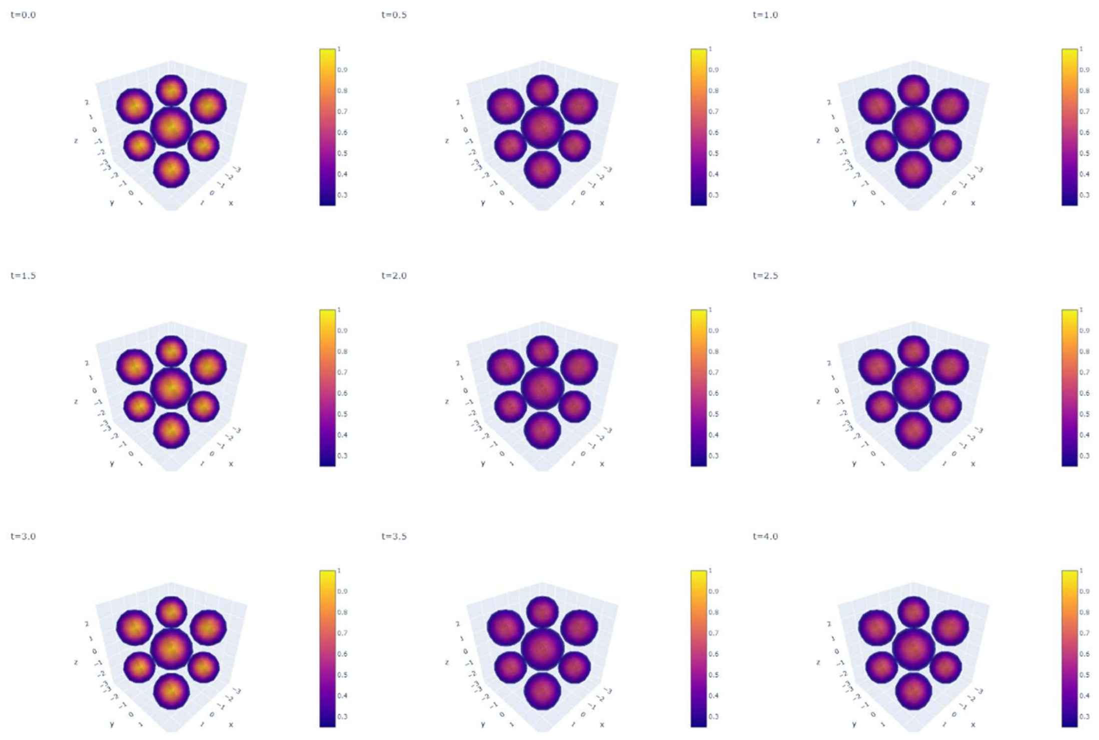
```python
# Plot Isosurfaces for Part A: Initial Condition:  ψ(x, y, z) = sin(x)*sin(y)*sin(z)
for i in range(len(t_eval)):
    image = (np.abs(np.fft.ifftn(sol2.y.T[i,:].reshape(n,n,n, order = 'F'))))**2
    fig = go.Figure(data=go.Isosurface(x=X.flatten(), y=Y.flatten(), z=Z.flatten(), value=image.flatten(),
                                       opacity=0.5, isomin=0.25, isomax=1, surface_count=20))
    fig.write_image(f"sin{t_eval[i]}.png")
```

Jonathan McCormack
12/10/2025
AMATH 581

Isosurfaces for Part A: Initial Condition: ψ(x,y,z) = cos(x)*cos(y)*cos(z)



Isosurfaces for Part B: Initial Condition: ψ(x,y,z) = sin(x)*sin(y)*sin(z)



Jonathan McCormack
12/10/2025
AMATH 581

Conclusion:

Bosons that start in initial conditions where the matter is condensed in small regions will be unaffected by periodic standing waves. This means that while the environment temperature remains sufficiently low, the Bose-Einstein Condensates at the ground quantum state will remain stable.

Implementation to Similar 3D Problems:

The code attached below is easily suited for any PDE that includes a combination of linear and nonlinear terms as long as the solution is evaluated with periodic boundaries centered at the spatial origin. The code is sufficiently general to solve increasing domain size, resolution, time span, or varying attraction/repulsion constants. This code's plotting function is also easily adopted to different use cases where the intent is to display regions of high or low density. However, the code is not easily modified to accommodate mixed boundary conditions.

More specifically, the Nonlinear Schrodinger equation serves as a way to model how waves interact with their environment. Real-world applications of this model include imaging or monitoring equipment that looks for specific regions of spectrum density of a specific type of particle. This could be used to track specific environmental contaminants, satellite/terrestrial radio waves, hospital nuclear imaging resonance, etc. Additionally, the Nonlinear Schrodinger equation is naturally very similar in formulation to many fluid, air, and climate dynamics equations.