Incompressible Schrodinger Flow

Robert Hafner
Computational Physics

December 20, 2024

1 Theory

To simulate fluid dynamics, two distinct methods are commonly used: the Lagrangian and Eulerian approaches. The Lagrangian method is considered a particle-based approach, where the positions and velocities of individual fluid particles or parcels are tracked over time. This method is particularly effective for simulating complex flows, such as vortex dynamics, and can handle complicated boundaries well, as it directly follows the motion of each fluid parcel. However, it can be computationally expensive, as it requires calculating the velocities and paths of a large number of particles, which can be time-consuming for flows with many particles. On the other hand, the Eulerian method is a grid-based approach, where a fixed grid is established, and velocity vectors are defined at each point in the grid. Fluid particles move through this grid, following the flow as dictated by the velocity field. While this method is computationally efficient and well-suited for large-scale fluid simulations, it is less effective for capturing complex phenomena like vortex dynamics and other intricate flow structures, as it doesn't track individual particles.
In 2016, Albert Chern, a phD student at the time, developed a grid based method that retains vortex dynamics in an elegant way. He penned this algorithm as incompressible schroedinger flow. This algorithm is based off of math formulated in 1926 by Erwin Madelung wrote the Schrodinger equation in a different form resembling hydrodynamics. The Schrodinger equation:

$$i\hbar\frac{\partial}{\partial t}\psi = -\frac{\hbar^2}{2}\Delta\psi + p\psi$$

Using $\psi = re^{i\theta}$ and $q = r^2$ which is the fluid density and $\mathbf{u} = \hbar\nabla\theta$ which is the fluid velocity. Using these arguments it can then be shown that Schrodingers equation, for a complex values ψ, can be written as:

$$\frac{\partial}{\partial t}q + \nabla\cdot(q\mathbf{u}) = 0$$

$$\frac{\partial}{\partial t}\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} = -\nabla(p - \frac{\hbar^2}{2}\frac{\Delta\sqrt{q}}{\sqrt{q}})$$

These are referred to as the quantum euler equations and represent incompressible and inviscid fluids. An issue with using a $\mathbb{C}$-valued wave-function is that the vorticity is trivial. This is a result of $\mathbf{u} = \hbar\nabla\theta$ in which the curl of a pure gradient is 0 so the vortex dynamics are not present in this equation. In Chern's incompressible schrodinger fluid he used a $\mathbb{C}^2$-valued wave-function $\Psi = [r_1e^{i\theta_1}, r_2e^{i\theta_2}]^\intercal$, and has the variables $q = r_1^2 + r_2^2 = \overline{\Psi}^\intercal\Psi$ and $\mathbf{u} = \frac{1}{q}\hbar\mathrm{Re}(-\overline{\Psi}^\intercal i\Delta\Psi)$, which results in

$$i\hbar\frac{\partial}{\partial t}\Psi = \frac{\hbar^2}{2}\Delta\Psi + p\Psi$$

$$\mathrm{Re}(\overline{\Psi}^\intercal i\Delta\Psi) = 0$$

This formulation provides a direct analog to classical Euler equations. The first equation parallels the pressure term $\frac{\partial}{\partial t}\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} = -\nabla p$, while the second equation corresponds to the incompressibility condition $\nabla\cdot\mathbf{u} = 0$. Using a $\mathbb{C}^2$ wave-function for fluid simulation, $\mathbf{u}$ is no longer a pure gradient. This results in the curl of the velocity being 0 resulting in a simulation that maintains vortex dynamics

2 Algorithm

In my simulation I will be using two different methods. The first, discussed above, would be a grid simulation using incompressible Schrodinger flow. The second, will be a classical simulation that uses the Lax–Friedrichs method to solve the hyperbolic PDE that is Eulers equation. In Chern's paper he uses MacCormack method as a comparison to his ISF simulation. I opted for the Lax–Friedrichs as it was discussed within the context of our class and can be applied here. The Lax–Friedrichs is more stable than the MacCormack method since it tends to be more dissipative, but Lax–Friedrichs is first order accurate whereas MacCormack is second order. The system I choose to simulate is quite basic so issues with significant dissipation and accuracy should not be relevant.
The set up for the ISF simulation has 2 classes: TorusDEC and a subclass ISF. Torus initialized the grid points and resolution with periodic indices. The first method defined in this class is the derivative of function which is just the derivative of a scalar field with periodic indices so it returns $v_x = \frac{f(x+\Delta x,y,z)}{\Delta x}$ with Δx being the grid indices that warp around at the boundary. The derivative of one form method is the differential of a 1 form to a 2 form. This can be thought of as a more generalized curl. It returns $w_x = \frac{v_y - v_y(x,y,z+\Delta z)}{\Delta y} + \frac{v_z - v_z(x,y+\Delta y,z)}{\Delta z}$ and w_y, w_z have similar structures. The derivative of a two form uses a finite difference to approximate the two form differential which returns $f = \frac{w_x(x+\Delta x,y,z)-w_x(x,y,z)}{(\Delta x)^2} + \frac{w_y(x,y+\Delta y,z)-w_y(x,y,z)}{(\Delta y)^2} + \frac{w_z(x,y,z+\Delta z)-w_z(x,y,z)}{(\Delta z)^2}$. There is the div method which is simply the divergence of the vector field and the Poisson solve which solves Poisson's equation in Fourier space using FFTs. The sharp and staggered sharp are both approximations of the gradient, the sharp operator takes averages of neighboring points where staggered sharp uses staggered grid points. In the standard grid the components of a vector field are stored at the same grid points whereas staggered are placed at different grid points, x say may be at the center of the grid cell but y and z could be at different locations in a grid cell. It is important to note that these methods can most likely be implemented using python modules such as numpy and scipy. I am not well versed in differential geometry so was unsure if the sharp operators, one-form and two form would exist in these modules. I opted to keep Chern's overall structure of his code while adjusting for indices and instances of dividing by 0 (which occurred in the Poisson solve). In the future I would like to use python imports due to the fact it would most likely result in improved simulation performance. This is due to the fact these modules, such as numpy, use multiprocessing which will improve performance.
The subclass of Torus is the ISF, incompressible Schrodinger flow. This is the crux of the fluid simulation. The basis of this class is that it constructs the Schrodinger mask operator which is $e^{i\lambda \frac{dt}{2}}$ where $\lambda = -4\pi^2\hbar(k_x^2 + k_y^2 + k_z^2)$ and is the eigenvalues of the Laplace operator. The Schrodinger mask operator is relevant as it simulates the time evolution of the wave-function. It is used in the Schrodinger Flow method which returns the manipulated wave-functions. ψ_1 and ψ_2 are in Fourier space and as such we use fftshift and conduct a Fourier transform to multiply it by the Schrodinger mask and use an inverse FFT after applying the mask operator. The resulting time integration returns the flow of the wave functions over time. The pressure projection method ensures the incompressibility using a the velocity method, and gauge transform methods with div and Poisson solve inherited from the Torus. The velocity of one form method takes the wave-functions and calculated the velocity field of the grid. The remaining methods used in the simulations is a normalizing method which just normalized the wave-functions and a gauge transform method.
The other subclass of TorusDEC is the Lax–Friedrichs class. This method has 3 methods: shift, advection, and enforce compressibility. The enforce compressibility method calculated through a pressure projection which is done by calculating the divergence of the field and solving a Poisson equation (using the inherited methods from Torus). From this, a pressure field is calculated, and then its gradient is subtracted from the velocity components. The next method is shift, this method just shifts to the neighboring index of the grid using np.roll. It could be done without the numpy method but would require for loops to continuously throughout the grid which is computationally very expensive. This is then used in the central method advection which is where the Lax-Friedrichs is used. The scalar field is updated using this equation:

$$\phi_{\text{new}} = \phi + \Delta t \left(-\frac{\partial \phi}{\partial x} v_x - \frac{\partial \phi}{\partial y} v_y - \frac{\partial \phi}{\partial z} v_z \right) + \text{Flux}_x + \text{Flux}_y + \text{Flux}_z - \text{Lax-Term}$$

Where

$$\text{Flux}_x = 0.5\left(\phi(x+\Delta x, y, z) + \phi(x-\Delta x, y, z)\right)\cdot v_x$$

and

$$\text{Lax-Term} = -0.5\Delta t\left(\frac{\phi(x+\Delta x,y,z)-\phi(x,y,z)}{\Delta x} + \frac{\phi(x,y+\Delta y,z)-\phi(x,y,z)}{\Delta y} + \frac{\phi(x,y,z+\Delta z)-\phi(x,y,z)}{\Delta z}\right)$$

The lax term is meant to handle any strong fluctuations such as shock-waves through introducing an artificial viscosity. In the case of this simulation, the lax term was found to be irrelevant as the simulation had no need for a dissipative term. However, I kept in case I want to try other simulations.
The final method is the particle class. This class generates an arbitrary number of particles to be moved through the velocity field using fourth order Runge-Kutta.

3 Results

The first check of the simulation was to advect particles in the velocity field. The videos for both ISF and Lax-Friedrich should be attached as GIFs. Visually speaking, both simulations are incredibly different. The Lax-Friedrich simulation of a jet flow in the x direction is quite dull, the particles only move through the x-direction with no vortex dynamics. The ISF simulation is much more rich as particles are moving forward in the x-direction but as they move through the simulation they are being shifted in the y and z direction. This occurs as particles slow down and are being moved to the side. We can also visualize this in a plot of the velocity field.

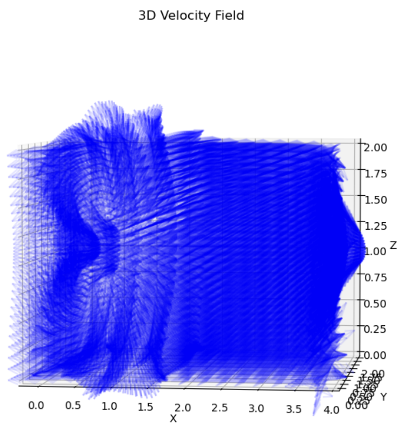

Figure 1: ISF velocity field

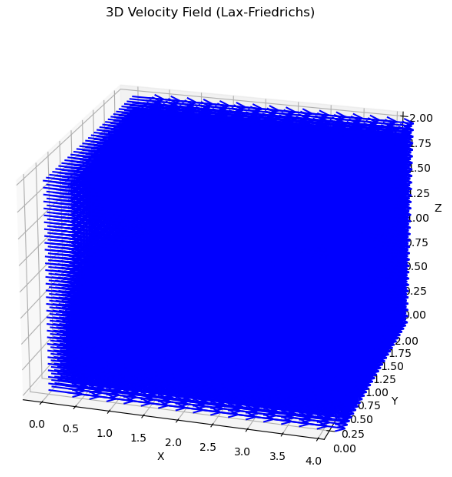

Figure 2: Lax-Friedrich velocity field

The Lax-Friedrich velocity is showing a uniform field moving only in the x-direction. The simulation has no vortex dynamics and is quite a poor simulation of the desired jet-flow. Comparing to the ISF field from a side view it can be seen that at the nozzle velocities are pointing to in the y and z direction. The resulting simulation of particles shows a swirling effect around the nozzle achieving the desired vortex dynamics not previously simulated in the Lax method. It can further be looked at through a plot of the vortex contours.

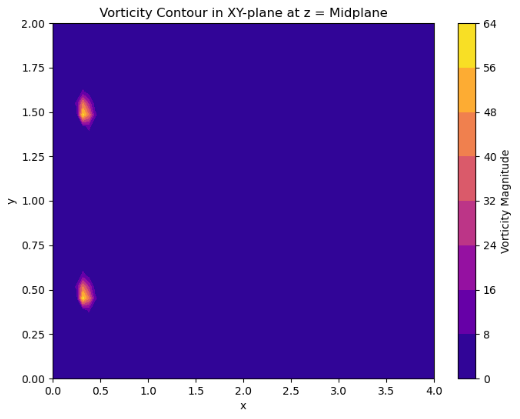

Figure 3: ISF vorticity contour

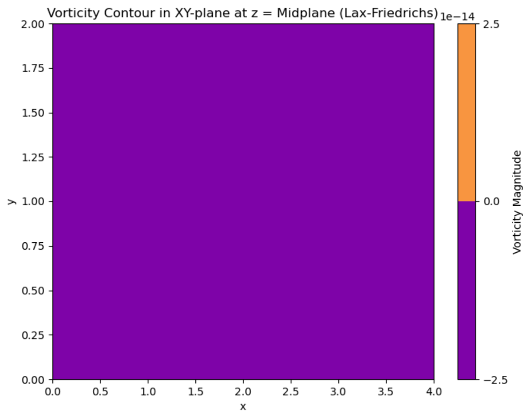

Figure 4: Lax-Friedrich vorticity contour

While this is a slice of the plot at the z-midplane it is seen that near the nozzle the vortices are formed and can be seen in the contour. The Lax simulation has no contour to speak as nothing is being changed and the flow is completely uniform. This can also be seen in the velocity contours

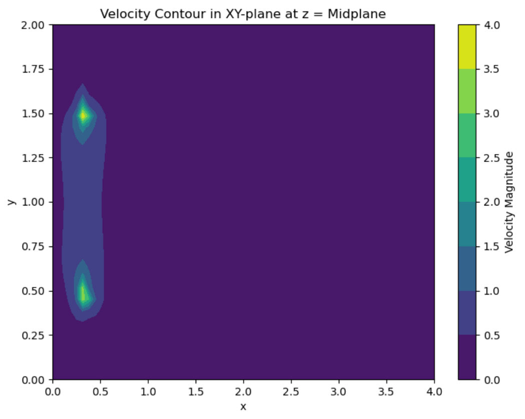

Figure 5: ISF velocity contour

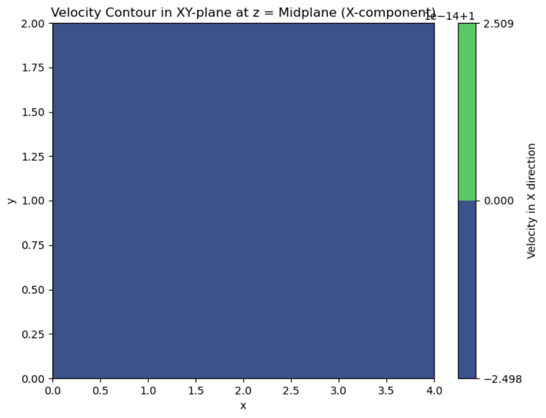

Figure 6: Lax-Friedrich velocity contour

Again, all the important information being lost in the Lax method is being lost at the nozzle. A lot of the interesting flows and changes in the velocity field are being depicted in ISF and lost entirely in the opposing grid method. This does not necessarily make ISF a vastly superior method since other methods such as MacCormack, semi-lagrangian, Lax-Wendroff are all different methods of solving pdes that could result in a simulation that better captures the vortex dynamics of a jet-flow. It could also be that the Lax-Friedrich method could be further enhanced to maintain the jet-flow. I would like to try and improve upon this grid implementation so a better comparison can be made. It would also be interesting create a boundary for the fluid to flow around. In the case of a purely Eulerian simulation it would be best to use Dirichlet boundary conditions in which the values along the boundary are fixed. In the case of ISF, Euler's equations are written as a $\mathbb{C}^2$ wave-function and as a result applying boundary conditions is not as trivial. I also decided to include plots of the phase of ψ_1 and the density of Ψ $(\psi_1^2 + \psi_2^2)$

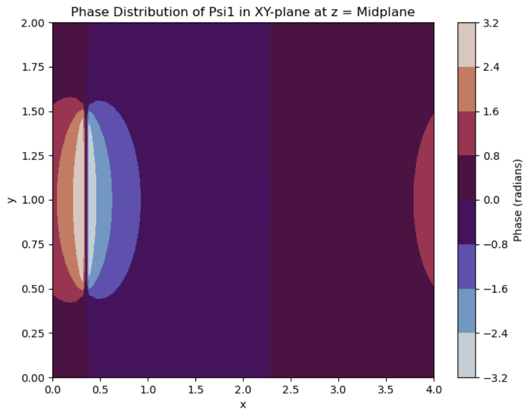

Figure 7: Phase Distribution ψ_1

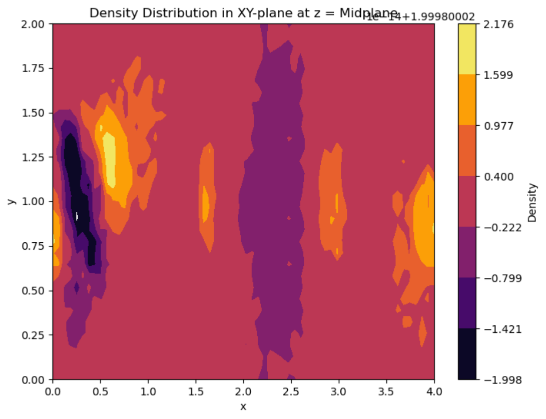

Figure 8: Density Distribution

4 References

[1] Chern, A., Knöppel, F., Pinkall, U., Schröder, P., & Weißmann, S. (2016). Schrödinger's smoke. ACM Transactions on Graphics, 35(4), 77. https://doi.org/10.1145/2897824.2925868
[2] A.Chern. (2017) Fluid Dynamics with Incompressible Schrödinger Flow. PhD Thesis, California Institute of Technology
[3] Riva, S., Introini, C., & Cammi, A. (2024). A finite element implementation of the incompressible Schrödinger flow method. Physics of Fluids, 36(1), 017138. https://doi.org/10.1063/5.0180356