CFDG VORTICITY-STREAMFUNCTION SOLVER JACOB IVANOV

1. Introduction

Oftentimes, the onboarding of a new researcher within computational fluids, or computational groups in general, is hampered by a lack of understanding on how to actually implement a method that is usually described in a page or two within a textbook. As a case in point, a basic description of the Voriticity-Streamfunction method is described in Section 7.2.4 of Ferziger's *Computational Methods for Fluid Dynamics*, and can be read in 10 minutes. However, implementing a Spectral variation of that method (as well as quite a few additions) has been the main focus of my research with Dr. Georgios Matheou within the Computational Fluid Dynamics Group (CFDG) over the course of the last year.

The goal of this documentation is to create an understandable guide to implementing this Spectral Vorticity-Streamfunction method, which another new researcher can follow and expand upon for themselves, and in the process gain familiarity with the basic requirements of computational implementations in general.

2. Spectral Vorticity-Streamfunction Method

2.1. **Governing Equations.** Starting from the very beginning, Issac Newton determined that $\vec{F} = m\vec{a}$, from which the 2D Navier-Stokes equations can be trivially derived[1]:

$$\frac{\partial u_x}{\partial t} + u_x\frac{\partial u_x}{\partial x} + u_y\frac{\partial u_x}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2}\right) \tag{1}$$

$$\frac{\partial u_y}{\partial t} + u_x\frac{\partial u_y}{\partial x} + u_y\frac{\partial u_y}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 u_y}{\partial x^2} + \frac{\partial^2 u_y}{\partial y^2}\right) \tag{2}$$

where the fluid velocity components are $\vec{u} = \langle u_x, u_y\rangle$, p is the fluid pressure, ν is the kinematic viscosity, and ρ is the fluid density. By differentiating (1) with respect to y, (2) with respect to x, and obtaining the difference, we can eliminate the pressure term, and obtain the following[2]:

$$\begin{aligned}\frac{\partial}{\partial t}\left[\frac{\partial u_x}{\partial y} - \frac{\partial u_y}{\partial x}\right] + (u_x+1)\frac{\partial^2 u_x}{\partial x\partial y} - (u_y+1)\frac{\partial^2 u_y}{\partial x\partial y} \\ + \frac{\partial u_y}{\partial y}\frac{\partial u_x}{\partial y} - \frac{\partial u_x}{\partial x}\frac{\partial u_y}{\partial x} + \nu\left(\frac{\partial}{\partial x}\left[\nabla^2 u_y\right] - \frac{\partial}{\partial y}\left[\nabla^2 u_x\right]\right) = 0\end{aligned} \tag{3}$$

From there, we can use the streamfunction ψ and vorticity ω as intermediate variables, which are defined below:

$$u_x = +\frac{\partial \psi}{\partial y}, \quad u_y = -\frac{\partial \psi}{\partial x} \tag{4}$$

[1]Left as an exercise to the reader. In all seriousness, this is very well explained in any introductory Fluid Mechanics textbook.

[2]Making the assumption of incompressiblity, i.e. $\rho = c$, $\nabla\cdot\vec{u} = 0$. Additionally, this derivation also works with some conservative body force $\vec{g}$ acting on the fluid, which has been ignored.

$$\omega = \nabla \times \vec{u} = \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y} = -\nabla^2 \psi \tag{5}$$

As a result, we can obtain the respective Streamfunction and Vorticity Transport formulations of the 2D Navier-Stokes equations.

$$\frac{\partial}{\partial t}\left[\nabla^2\psi\right] + \frac{\partial \psi}{\partial y}\frac{\partial}{\partial x}\left[\nabla^2\psi\right] - \frac{\partial \psi}{\partial x}\frac{\partial}{\partial y}\left[\nabla^2\psi\right] = \nu\nabla^2\left[\nabla^2\psi\right] \tag{6}$$

$$\frac{\partial \omega}{\partial t} + u_x\frac{\partial \omega}{\partial x} + u_y\frac{\partial \omega}{\partial y} = \nu\nabla^2\omega \tag{7}$$

Due to it's compactness, we will use Eq. (7) going forward. Additionally, though this method is referred to as the Vorticity-Streamfunction method, it is not necessary to calculate the Streamfunction intermediately.

2.2. **Spectral Vorticity Transport Equation.** Within this solver, we will be using spectral methods to time-step and obtain derivatives, and as a result, we need to rewrite Eqs. (5) and (7).

$$\frac{\partial \Omega}{\partial t} + \nu(k_p^2 + k_q^2)\Omega = -\mathcal{F}_{+2}\left[u_x\frac{\partial \omega}{\partial x} + u_y\frac{\partial \omega}{\partial y}\right] \tag{8}$$

$$u_x = \mathcal{F}_{-2}\left[U_x\right] = \mathcal{F}_{-2}\left[+\frac{\underline{i}k_q\Omega}{k_p^2 + k_q^2}\right], \quad u_y = \mathcal{F}_{-2}\left[U_y\right] = \mathcal{F}_{-2}\left[-\frac{\underline{i}k_p\Omega}{k_p^2 + k_q^2}\right] \tag{9}$$

$$\frac{\partial \omega}{\partial x} = \mathcal{F}_{-2}\left[\underline{i}k_p\Omega\right], \quad \frac{\partial \omega}{\partial y} = \mathcal{F}_{-2}\left[\underline{i}k_q\Omega\right] \tag{10}$$

where $\underline{i}$ is the imaginary unit, k_p and k_q are the spectral x- and y-derivative operators, Ω, U_x, and U_y are the frequency version of ω, u_x, and u_y, respectively, and $\mathcal{F}_{+2}$, $\mathcal{F}_{-2}$ are the forward and backward 2D Fourier Transforms, respectively. Unfortunately, we need to repeatedly use the Fourier Transform between physical/frequency space to compute the non-linear product in the convection term.

Time-stepping Eqs. (8)-(10) is possible, but due to the combination of first- and second-order derivatives, numerical stability is difficult to maintain. As a result, we can further modify the Spectral Vorticity Transport Equation to use an integrating factor:

$$\Xi(t_n) = \exp\left[\nu(k_p^2 + k_q^2)t_n\right] \tag{11}$$

$$\frac{\partial}{\partial t_n}\left[\Xi(t_n)\cdot\Omega\right] = -\underline{i}\Xi(t_n)\cdot\mathcal{F}_{+2}\left[u_x\frac{\partial \omega}{\partial x} + u_y\frac{\partial \omega}{\partial y}\right] \tag{12}$$

Unfortunately, because $\partial\Xi/\partial t$ is strictly positive, as t_n increases, the $\Xi(t_n)$ variable will become gradually less precise, and eventually will overflow due to the floating point format it is stored as computationally. As a result, we can make our final modification[3] to our Spectral Vorticity Transport equation:

$$\frac{\partial}{\partial t_n}\left[\Xi(c_s\Delta t)\cdot\Omega\right] = -\underline{i}\Xi(c_s\Delta t)\cdot\mathcal{F}_{+2}\left[u_x\frac{\partial \omega}{\partial x} + u_y\frac{\partial \omega}{\partial y}\right] \tag{13}$$

where c_s is the time supstep fraction for the Runge-Kutta integration method described later, and Δt is the time-step.

[3]This change is legitimate because it is equivalent to 'reseting' our $\Xi(t_n)\Omega$ variable by pre-multiplying by $\Xi(-t_n)$ at the beginning of each integration timestep.

2.3. **Initial Condition.** While any smooth 2π-periodic initial velocity field would work, Taylor-Green flow is a natural choice, and gives us the ability to test against the analytical solution over time. Taylor-Green flow is described by:

$$\begin{cases} u_x(t,x,y) = +e^{-2\nu t}\cos(\beta x)\sin(\beta y) \\ u_y(t,x,y) = -e^{-2\nu t}\sin(\beta x)\cos(\beta x) \end{cases} \tag{14}$$

where β is a scaling integer constant. The vortex wavelength will be $2\pi/\beta$. However, any perturbation to the initial condition will cause the flow to destabilize, resulting in turbulence. As a result, we will set the initial condition instead to the following:

$$\begin{cases} u_x(t=0,x_i,y_j) = +\cos(\beta x_i)\sin(\beta y_j) + \gamma_x \\ u_y(t=0,x_i,y_j) = -\sin(\beta x_i)\cos(\beta y_j) + \gamma_y \end{cases} \tag{15}$$

where γ_x and γ_y represent uniform `rng`-generated perturbations to the x- and y-velocity. Its magnitude must be relatively small, and has been set to 0.02 for this project.

2.4. **Runge-Kutta Integration Method.** Now that we have the final form of our governing equations, as well as an initial condition, we need a way to integrate them over time. One might immediately think of something like Euler's Method, where you can timestep as follows:

$$y_{n+1} = y_n + \Delta t \cdot f(t_n, y_n) \tag{16}$$

where $f = \mathrm{d}y/\mathrm{d}t$. Euler's Method is actually a first-order version of a Runge-Kutta (RK) method, but is unstable for typical fluid dynamics problems[4], and as a result, we will need to use a higher order RK method. In general[5], RK methods are described by three coefficient matrices: **a**, **b**, and **c**, which are referred to as the RK Matrix, Weights Matrix, and Nodes Matrix. The solution for the next time-step can be constructed as follows:

$$\begin{cases} y_{n+1} = y_n + \Delta t \sum_{i=1}^{s} [b_i k_i] \\ k_i = f\left(t_n + c_i \Delta t, y_n + \Delta t \sum_{j=1}^{i-1} [a_{ij} k_j]\right) \end{cases} \tag{17}$$

We will specifically use Wray's Method[6], with a Butcher Table as follows:

$$\begin{array}{c|c} \mathbf{c} & \mathbf{a} \\ \hline & \mathbf{b} \end{array} = \begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 8/15 & 8/15 & 0 & 0 \\ 2/3 & 1/4 & 5/12 & 0 \\ \hline & 1/4 & 0 & 3/4 \end{array} \tag{18}$$

[4]Consult Chapter 7: "Stability of Linear Systems" within Lomax's *Fundamentals of Computational Fluid Dynamics* for a full explanation.

[5]There are more advanced variations of RK Methods, ranging from implicit, adaptive, etc. that are descibed well in the Wikipedia Article on RK Methods. Within this project, we have only done explicit forms.

[6]Also referred to as Van der Houwen's Method. The autonomous version of which was described within *Spectral Methods for the Navier-Stoes Equations with One Infinite and Two Periodic Directions* by Spalart et al.

However, we cannot pick any arbitrary Δt and maintain stability, and so we use the Courant-Friedrich-Lewy (CFL) condition. Prior to starting an RK3 step, we find the $\max[u_x]$ and $\max[u_y]$, and find Δt as follows:

$$\Delta t = \frac{\Delta x \Delta y}{\Delta y \max[u_x] + \Delta x \max[u_y]} \tag{19}$$

2.5. **Particle Transport.** Consider a point particle dropped into a flow. If the particle is non-inertial, it will instantaneouly be travelling at the same velocity and direction as the flow surrounding it.

$$\vec{v}(t, \vec{x}_p) = \vec{u}(t, \vec{x}_p) \tag{20}$$

where $\vec{x}$ is the particle position vector, $\vec{u}$ is fluid velocity vector, and $\vec{v}$ is the particle velocity vector. As a result, the particle position can be directly time-stepped by updating the particle position by its velocity. However, if the particle is inertial, the particle velocity and fluid velocity will no longer be instantaneously equal, but rather, will be described by the following:

$$\frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \frac{1}{\tau}(\vec{u} - \vec{v}) + 3\vec{a} \tag{21}$$

where $\vec{a}$ is the fluid acceleration[7], and τ is the non-dimensional inertial time[8]. Due to the coupled nature of Eq. (20), the following Predictor-Corrector (PC) scheme[9] was implemented:

Predictor:

$$\vec{x}_{n+\alpha} = \vec{x}_n + \Delta t \cdot \vec{v}_n \tag{22}$$

$$\vec{u}_{n+\alpha} = \vec{u}(t_n, \vec{x}_{n+\alpha}) \tag{23}$$

$$\vec{v}_{n+\alpha} = \frac{\vec{v}_n + \Delta t \left[\dfrac{\vec{u}_{n+\alpha}}{\tau} + 3\vec{a}_n \right]}{1 + \dfrac{\Delta t}{\tau}} \tag{24}$$

Corrector:

$$\vec{x}_{n+1} = \vec{x} + \frac{\Delta t}{2}(\vec{v}_n + \vec{v}_{n+\alpha}) \tag{25}$$

$$\vec{u}_{n+1} = \vec{u}(t_{n+1}, \vec{x}_{n+1}) \tag{26}$$

$$\vec{a}_{n+1} = \vec{a}(t_{n+1}, \vec{x}_{n+1}) \tag{27}$$

$$\vec{v}_{n+1} = \frac{1}{1 + \frac{\Delta t}{2\tau}} \left[\vec{v}_n + \frac{\Delta t}{2} \left(3(\vec{a}_n + \vec{a}_{n+1}) + \frac{\vec{u}_n + \vec{u}_{n+1} - \vec{v}_n}{\tau} \right) \right] \tag{28}$$

[7]It should be noted that particle transport is by default Lagrangian rather than Eulerian, and as a result, the acceleration experienced by a fluid element at position $\vec{x}$ is $\vec{a} = \mathrm{D}\vec{u}/\mathrm{D}t \neq \mathrm{d}\vec{u}/\mathrm{d}t$, or more explicitly:

$$\vec{a}(\vec{x}) = \frac{\mathrm{D}\vec{u}}{\mathrm{D}t} = \frac{\mathrm{d}\vec{u}}{\mathrm{d}t} + (\nabla \cdot \vec{u}(\vec{x}))\vec{u}(\vec{x})$$

If this is unintuitive to you, consider a steady, but non-uniform fluid velocity field $\vec{u}(\vec{x})$. Since it is steady, $\mathrm{d}\vec{u}/\mathrm{d}t = 0$, but is the fluid acceleration zero everywhere?

[8]The inertial time τ is dependent on the particle geometry, fluid viscosity, particle/fluid density ratio, etc. but here will be assumed to be just an independent constant.

[9]This scheme is from *Effect of Bubble Size on Lagrangian Pressure Statistics in Homogeneous Isotropic Turbulence* by M.H. Bappy et al., but excludes bubble buoyancy.

It can be seen that in the non-inertial case, where $\tau = 0$, this PC scheme degenerates to Heun's Method.

2.6. **Spectral Interpolation.** Ironically, though described last in the this project documentation's methods, this was the first aspect of the project that was completed. In order to obtain accurate interpolated values for fluid velocity, acceleration, etc. at locations in between the grid points, we can exploit the periodic nature of the flow and use Spectral Interpolation, where we rebuild the point's value by using the Fourier Coefficients.

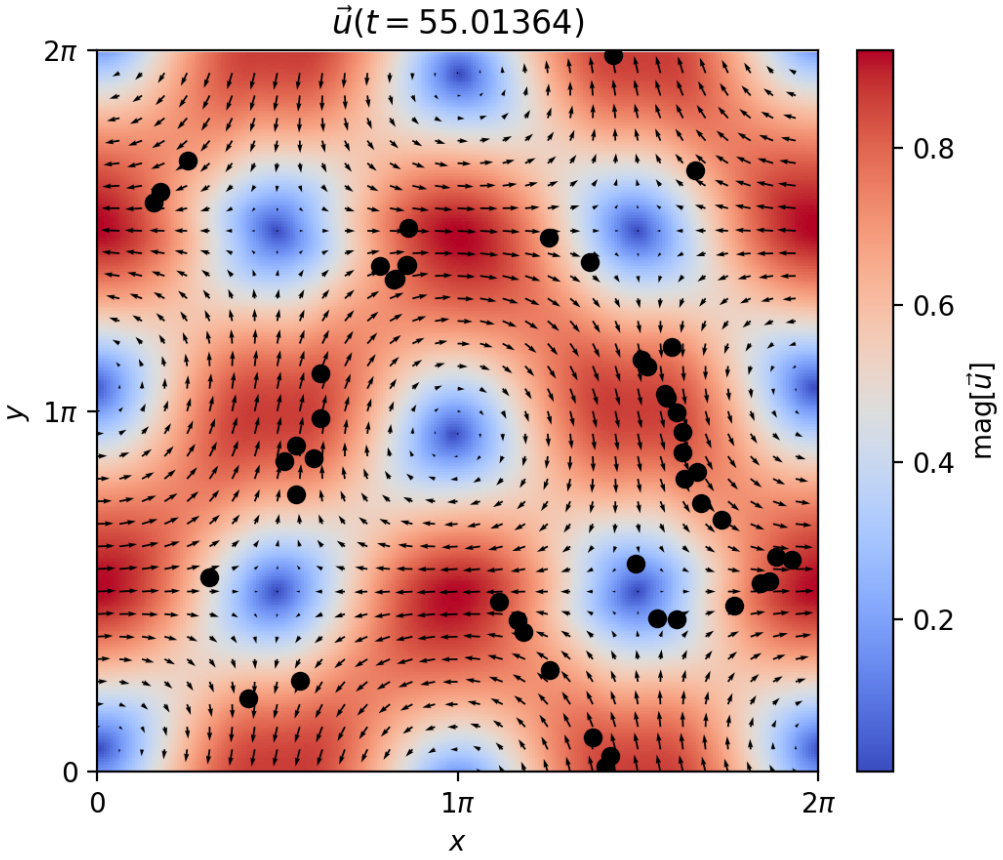
Assuming the values of a discrete periodic function $y_i(x_i)$ on domain $x_i \in [0, 2\pi)$, the Fourier Transform of y_i is $Y_p = \mathcal{F}_{+1}[y_i]$. An interpolating polynomial can be constructed as:

$$p(x) = \frac{1}{N} \sum_p \left[Y_p e^{\underline{i} k_p x} \right] \tag{29}$$

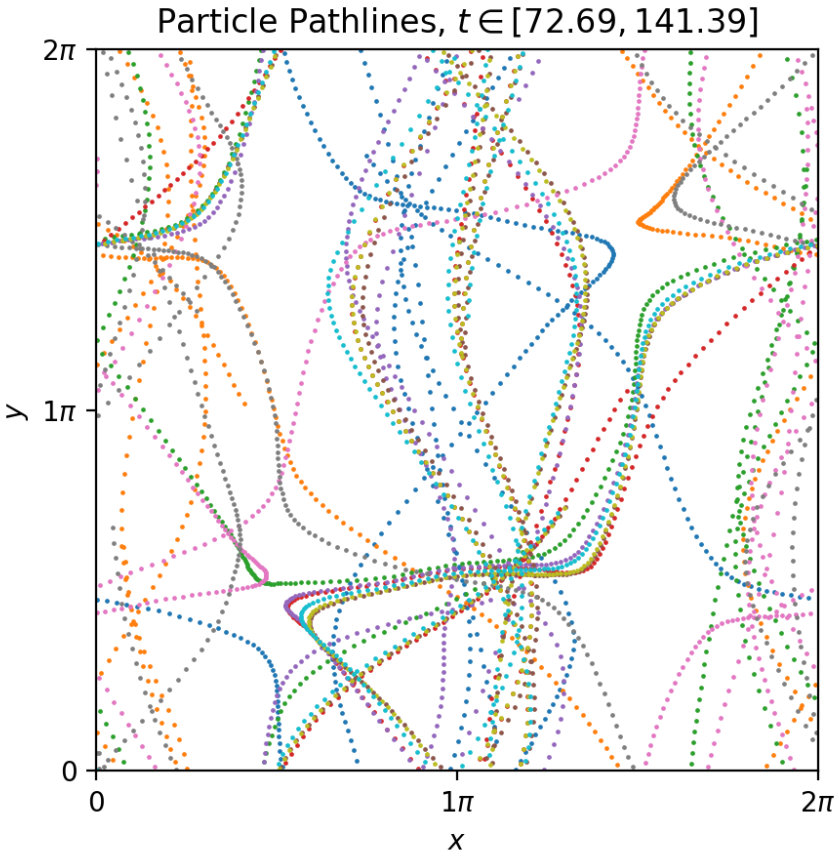
$p(x)$ can be evaluated at the target location in order to find the interpolated value at the location. In higher dimensions, this will need to repeated across all x, then across a single y, etc.

3. Results

In total, we have a Spectral Voriticity-Streamfunction method Navier-Stokes solver with inertial particle transport. The script `vorticity_streamfunction_solver.py` can run to obtain similar results:



By tracking each particle for the time period following destabilization, once turbulence is initiated, it can be seen that these particles follow independent and chaotic paths.



4. Verification Tests

4.1. **Verification of Spectral Differentiation.** The first thing we need to verifiy is our ability to spectrally differentiate and anti-differentiate.

The Fourier Transform is defined as the following:

$$u_i = \sum_i \left[U_p \cdot e^{\underline{i} k_p x_i} \right] \tag{30}$$

The derivative can be found as follows:

$$\frac{\mathrm{d}u_i}{\mathrm{d}x} = \sum_i \left[\underline{i} k_p U_p \cdot e^{\underline{i} k_p x_i} \right] = \underline{i} k_p \sum_i \left[U_p \cdot e^{\underline{i} k_p x_i} \right] \tag{31}$$

As a result, we can see that the spectral derivative operator equivalent for $\mathrm{d}/\mathrm{d}x = \underline{i} k_p$. Unfortunately, the exact ordering of the wavenumbers k_p is dependent on the implementation of the Discrete Fourier Transform used.

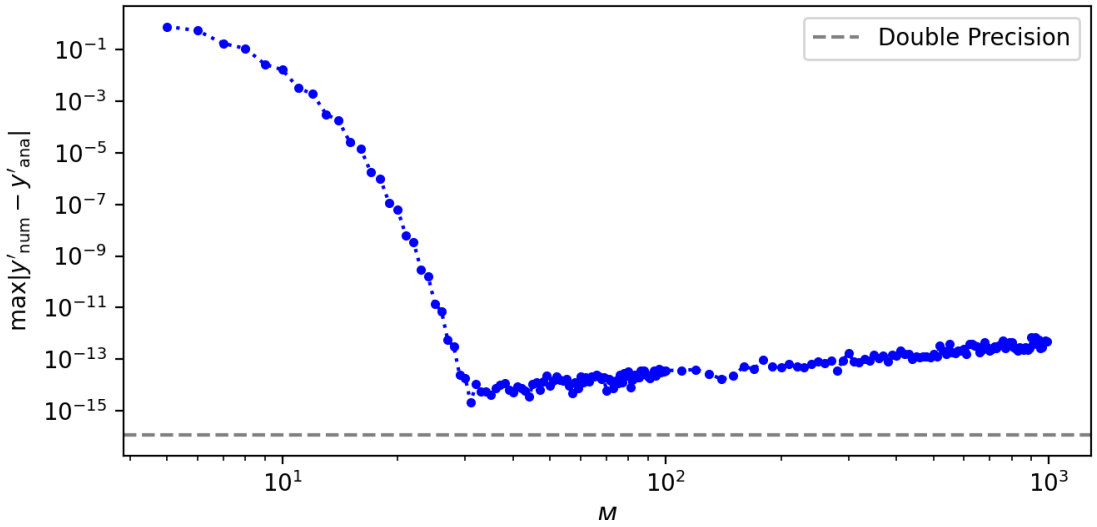
Throughout this entire project, development was done with Python, and specfically used the `numpy.fft` package[10], which uses the following format:

[10] `rocket-fft` was used to make `numba.njit` aware of `numpy.fft` functionality. It must be installed, or JIT-compilation disabled within `vorticity_streamfunction_solver.py` or an error will result.

```
k = [0, 1, ...,   M/2-1,     -M/2, ..., -1]   if M is even
k = [0, 1, ..., (M-1)/2, -(M-1)/2, ..., -1]   if M is odd
```

where `M` is the number of grid divisions in the range $x \in [0, 2\pi)$. An arbitrary periodic formula was chosen, $y = \exp[\cos(x)]\sin(x)$, with an analytical derivative $y' = -\exp[\cos(x)]\left[\sin^2(x) - \cos(x)\right]$, and a convergence test was completed by increasing the grid spacing and finding the absolute error between the spectrally differentiated and analytical y'. Figure 1 can be reproduced by executing the script `spectral_derivative.py` (runtime: < 1 s).

Figure 1: Convergence of Spectral and Analytical Derivative



4.2. **Verification of Spectral Poisson Equation.** Now that we have verified our ability to take spectral derivatives, we can tackle the slightly more challenging spectral Poisson Problem. As stated previously, $\omega = -\nabla^2\psi$. As a result, the frequency versions of voriticity and streamfunctions are related as follows in 2D:

$$\Omega_{pq} = -\left[-1k_p^2\Psi_{pq} - 1k_q^2\Psi_{pq}\right] \tag{32}$$

$$\Psi_{pq} = \frac{\Omega_{pq}}{k_p^2 + k_q^2} \tag{33}$$

However, when $p = q = 0$, Ψ_{00} is of form $1/0$. As a result, we are forced to ignore the value of Ω_{00}. As a result, our recovered ψ_{num} will have an average of 0, and we will need to correct by the average of the analytical. This is not a problem for our overall solver becasue we never actually need the value of ψ, only the derivatives.

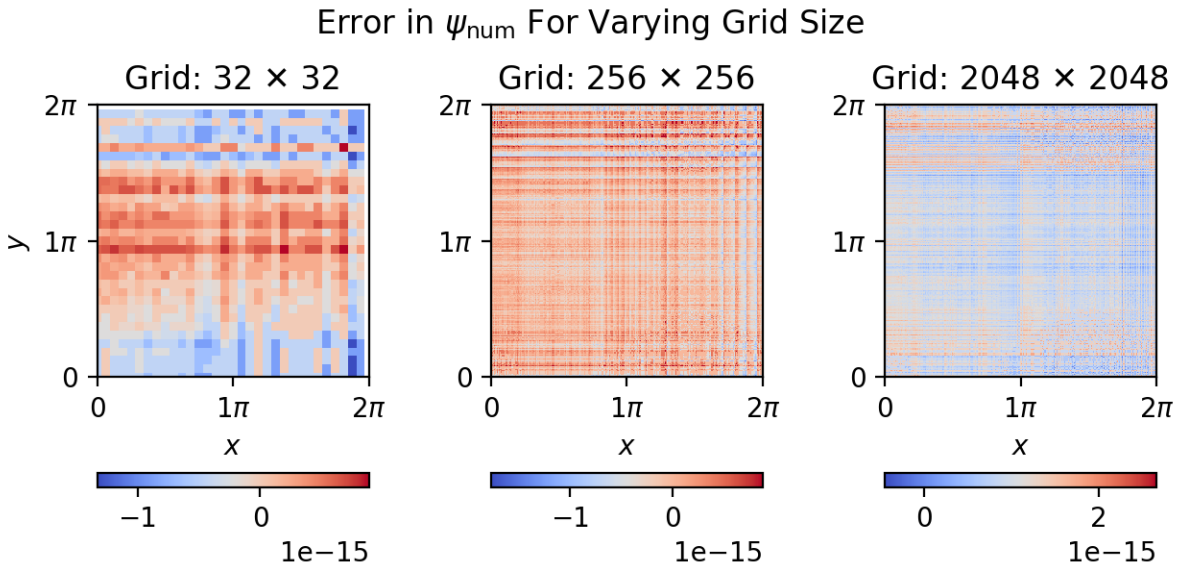
$$\psi_{\text{num}} - \bar{\psi}_{\text{ana}} = \mathcal{F}_{-2}\left(\frac{\Omega_{pq}}{k_p^2 + k_q^2}\right) \tag{34}$$

Again, we will pick an arbitrary analyitical function for $\psi = \exp[\cos(x)] - \sin(y)$ from which we can find a consistent $\omega = e^{\cos(x)}\left[\cos(x) - \sin^2(x)\right] - \sin(y)$. A convergence test using an ℓ_p-norm[11], can be completed, yielding results similar to Figure 1, but below the simple error

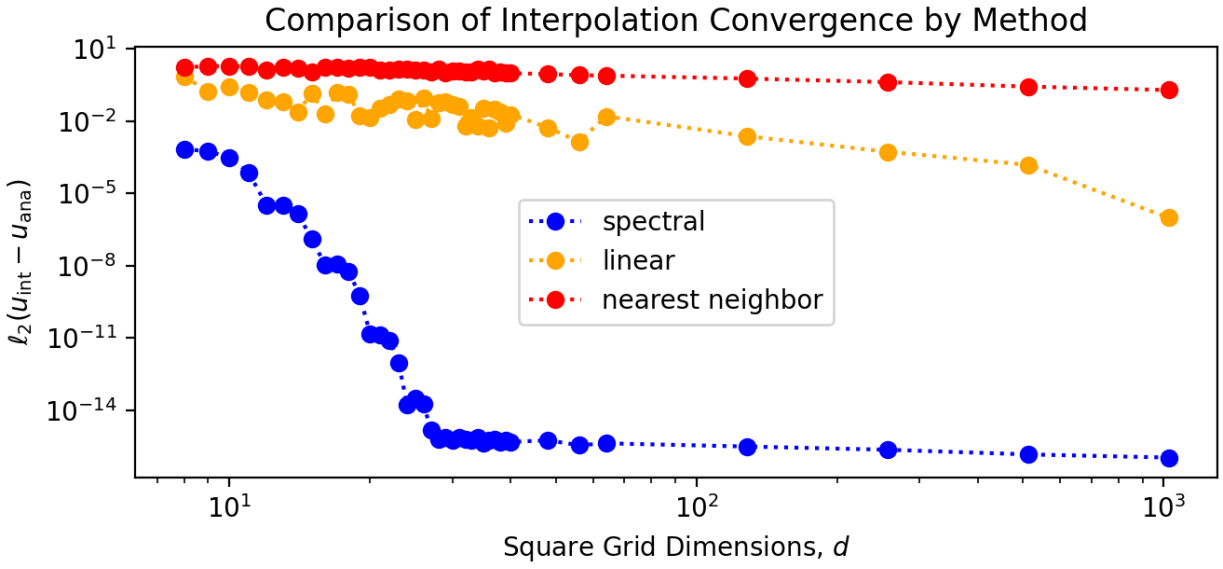
[11]In order to quantify the error in a multidimensional matrix, an ℓ_p-norm is used, defined as:

$$\ell_p[\mathbf{e}] = \left[\Delta x \Delta y \Delta z \sum_i \sum_j \sum_k \left(e_{ijk}^p\right)\right]^{1/p}$$

field is plotted for varying computational mesh sizes. Similar to Figure 1, because the lowest mesh shown is finer than ≈ 30, the error is on the order of machine epsilon already. Figure 2 can be reproduced by executing the script `spectral_poisson.py` (runtime < 5 s).



4.3. **Verification of Spectral Interpolation.** In order to verify our Spectral Interpolation method, we will compare the analytical values of the function $f = \exp[\sin(x) + \sin(y)]$ to our interpolation. This was accomplished in the script `inter2D_convergence.py` (runtime < 3 m).



4.4. **Verification of RK3 Method.** Confident that we have the required ability to compute spectral derivatives and anti-derivatives, and thereby the spectral Poisson equation, we can

which is straightforward to simplify for lower dimensions.

move on to testing our timestepping mechanism. As was stated previously, we will be using Wray's RK3 method, which after expanding out the Butcher table, cooresponds to

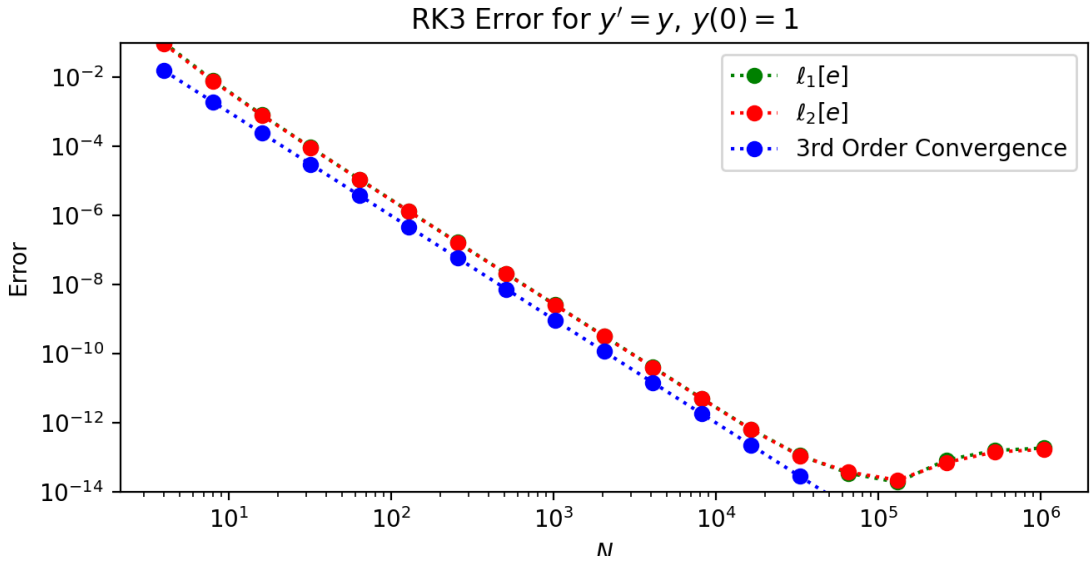
$$k_1 = f(t_n, y_n) \tag{35}$$

$$k_2 = f\left(t_n + \frac{8}{15}\Delta t, y_n + \Delta t \cdot \frac{8}{15}k_1\right) \tag{36}$$

$$k_3 = f\left(t_n + \frac{2}{3}\Delta t, y_n + \Delta t \cdot \left(\frac{1}{4}k_1 + \frac{5}{12}k_2\right)\right) \tag{37}$$

$$y_{n+1} = y_n + \Delta t\left[\frac{1}{4}k_1 + \frac{3}{4}k_3\right] \tag{38}$$

As before, we find an analytical problem that we can solve, and test against the known solution. The autonomous[12] equation $y' = y$ has the known solution $y(t) = y(0) \cdot e^t$.



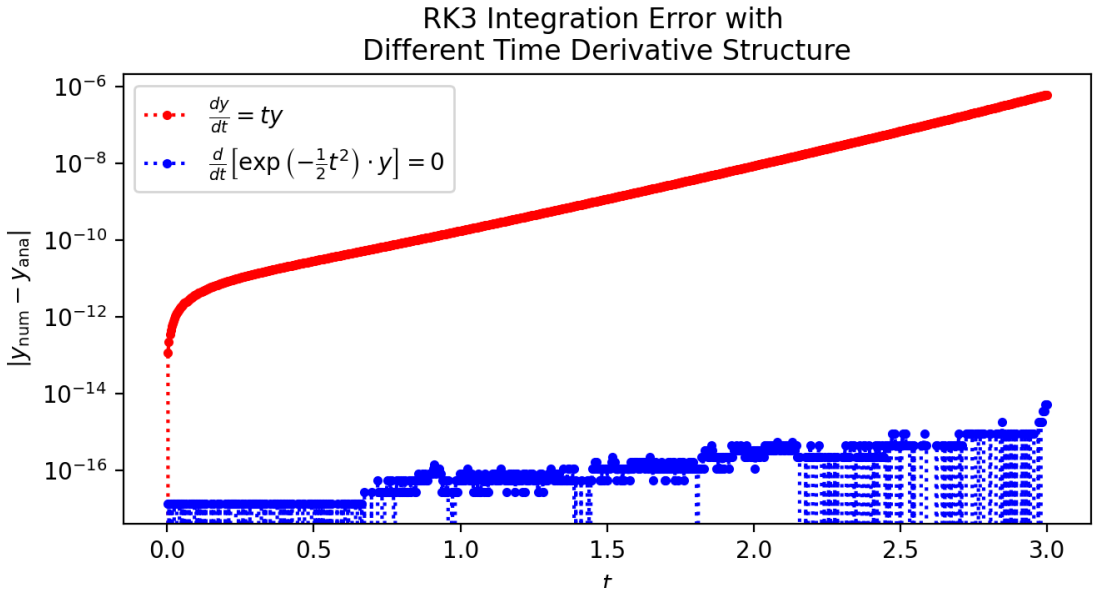
This figure can be reproduced by executing the script `rk3_exp.py` (runtime < 1 s).

Interestingly enough, modifying the form of f can dramatically change the error in the RK integation. The following are equivalent:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = ty \iff \frac{\mathrm{d}}{\mathrm{d}t}\left[\exp\left(-\frac{1}{2}t^2\right) \cdot y\right] = 0 \tag{39}$$

But integrating each form shows dramatically different error:

[12] An autonomous ODE's solution is independent of when the initial condition is applied. The RHS has no t-dependence.



This figure can be reproduced by executing the script `rk3_error_f_structure.py` (runtime < 1 s).

4.5. **Verification of Model Advection PDE.** The Advection Equation is a standard model PDE of advection-type problems, which is described by the following.

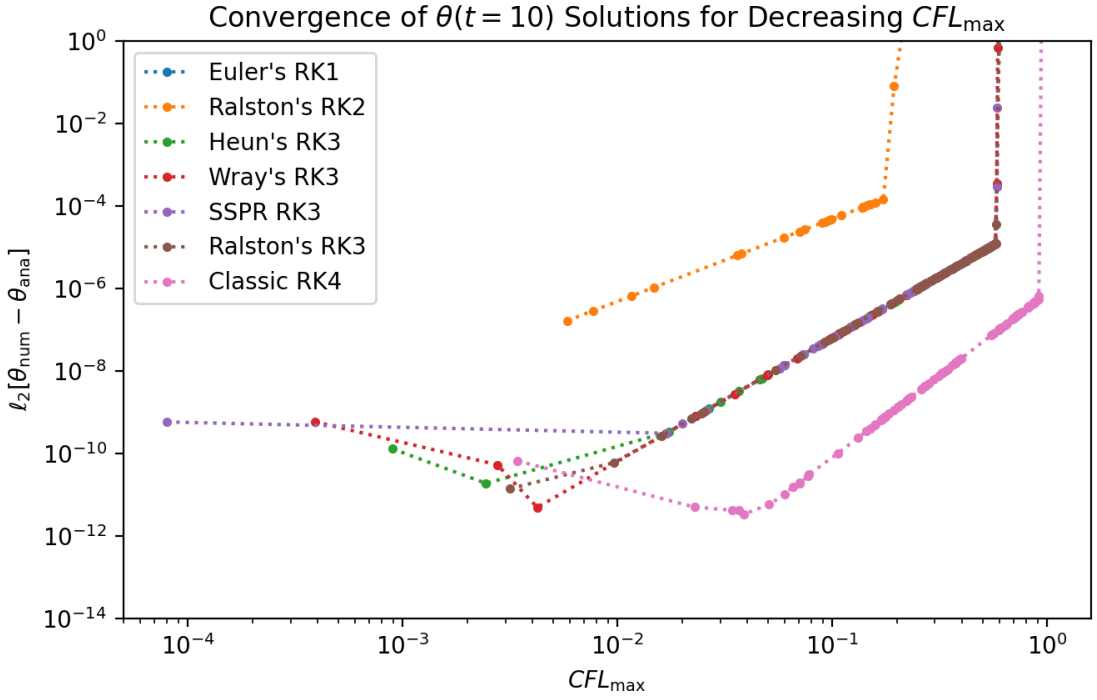
$$\frac{\partial \theta}{\partial t} + a\frac{\partial \theta}{\partial x} = 0 \tag{40}$$

where a is a constant fluid velocity, and θ represents some scalar transport variable, i.e. temperature, but note that this would represent no diffusivity. While it can be naturally extended to higher dimensions, we will first consider the 1D case, which has the analytical solution[13]:

$$\theta(t, x) = \theta(0, x - at) \tag{41}$$

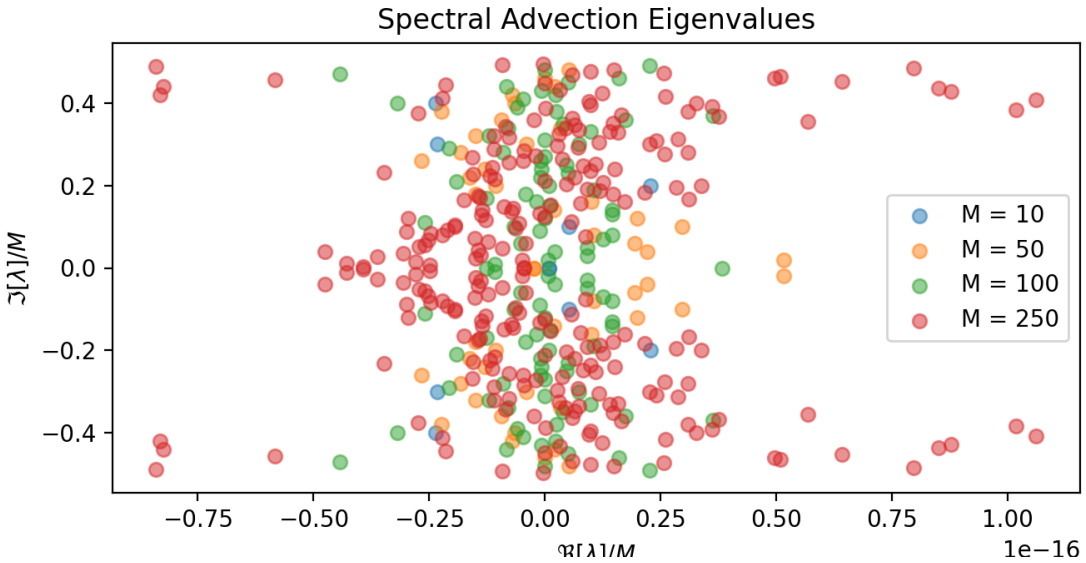
We will show that with decreasing Δt, with a maximum prescribed by the CFL condition, that the error between the analytical and numerical solution goes to zero, with a variety of methods.

[13]Solving first-order linear PDEs such as this one can often be accomplished by utilizing the Method of Characteristics.



It should be noted that Euler's RK1 and Ralston's RK2 are unstable for the advection equation, and integrating up to a higher time will show that for RK2 as well. The above figure can be replicated by executing `advection_error.py` (runtime < 5 m).

In order to determine the minimum RK scheme that will be stable, the eigenvalues of the differentiation matrix need to be found, and compared to the stability contours of RK schemes. For this advection problem (spectral or finite difference), the eigenvalues will be found solely on the imaginary axis in the complex plane.



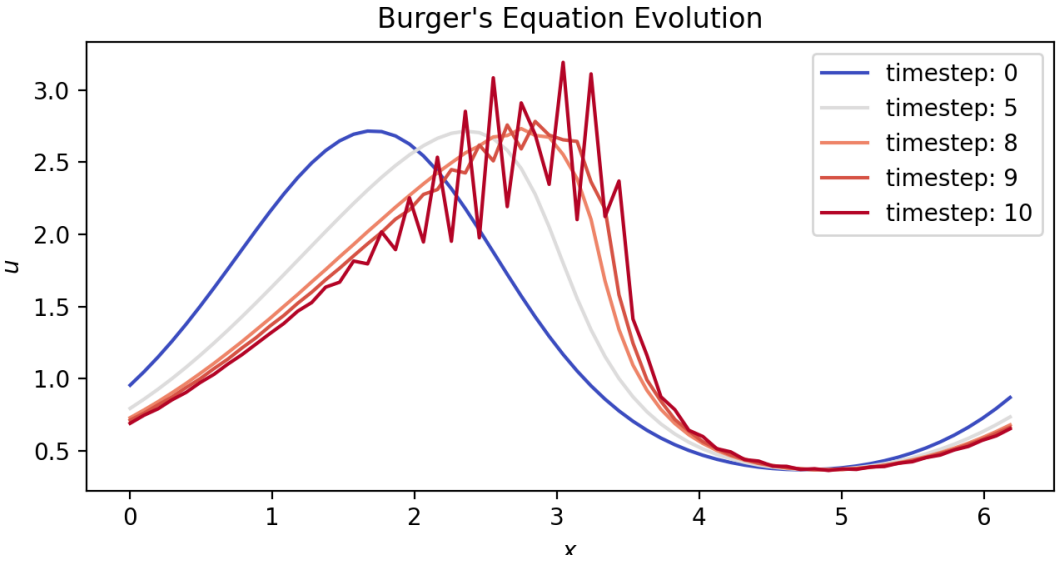
The derivation of the spectral differentiation matrix is not trivial. However, the above figure can be reproduced by executing `spectral_eigenvalues.py` (runtime < 1 s).

4.6. **Sandbox Test of Shock Evolution.** Though not directly relevent to the developement of the Voriticity-Streamfunction solver, shock evolution (and supersonic flow in general) were of particular interest to the developer. Burger's Equation, is a standard model PDE for shock-type problems.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial \left[u^2\right]}{\partial x} = 0 \tag{42}$$

which are this equation's advective and conservative forms, respectively.

An explicit RK3 time-stepped, spectral-space solution of these two forms is almost identical, with the Advective form having very slightly less upwind oscillation. Regardless, both exhibit numerical instability and are not able evolve to a full shock. The following plot can be reproduced with `shock_evolution.py` (runtime < 5 s).



4.7. **Verification of Model Diffusion PDE.** The Heat Equation is a model PDE for diffusion-type problems, which the following form:

$$\frac{\partial \theta}{\partial t} = \nu \frac{\partial^2 \theta}{\partial x^2} \tag{43}$$

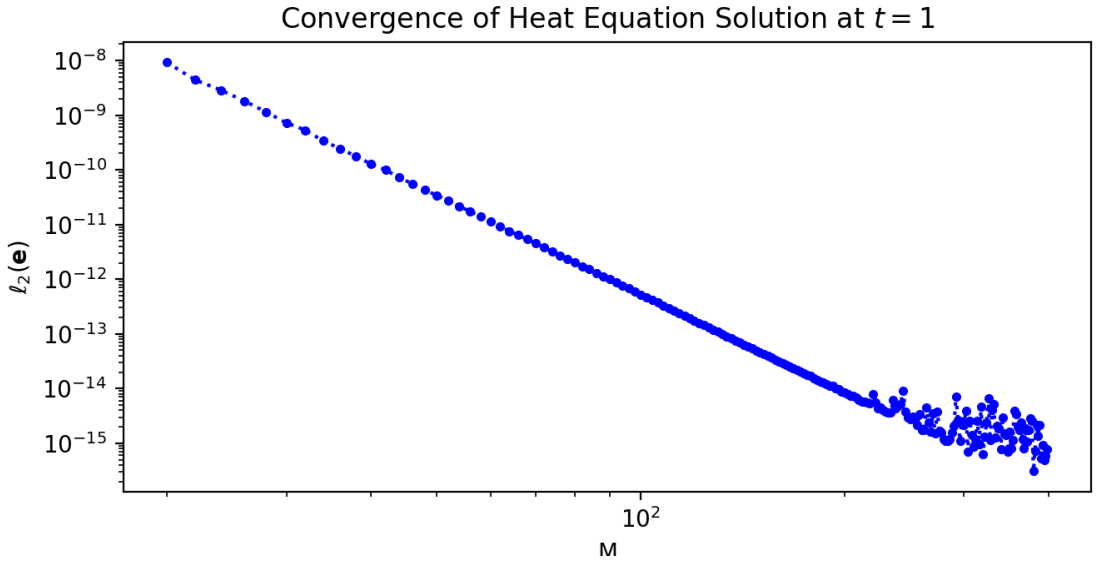
By assuming an initial condition with an analytical solution[14], we can perform a convergence test to verify we can numerically solve this type of problem. Choosing an arbitrary initial condition:

$$\theta(t = 0, x) = \sin(ax) \tag{44}$$

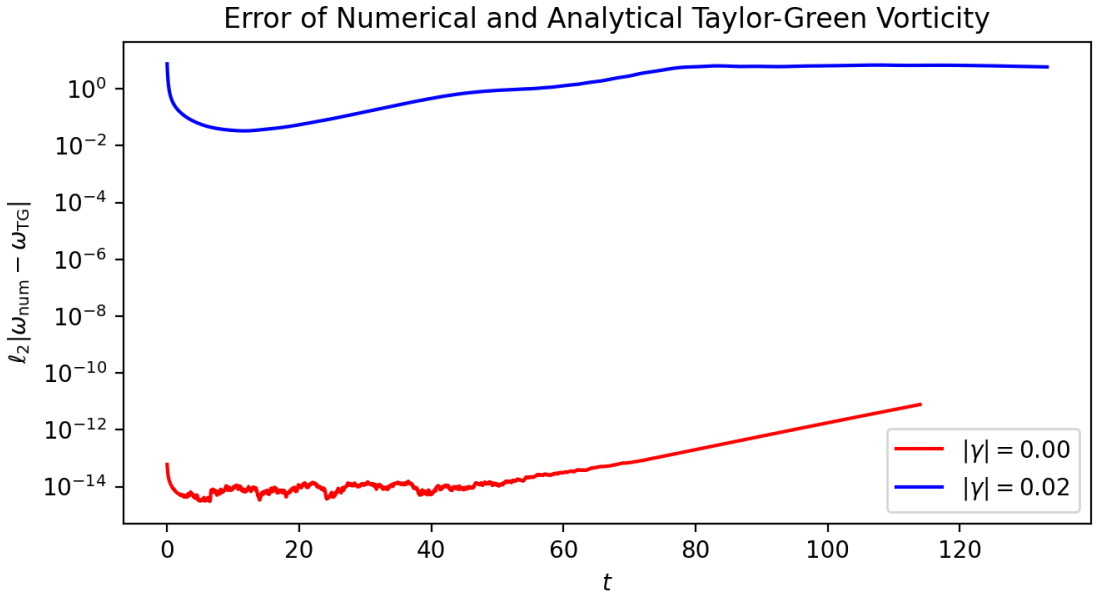
$$\theta(t, x) = e^{-\nu a^2 t} \sin(ax) \tag{45}$$

The convergence test performs as expected. The following plot can be reproduced by running `heat_error.py` (runtime < 5 m).

[14]The most common way to solve this type of PDE is to use Seperation of Variables.



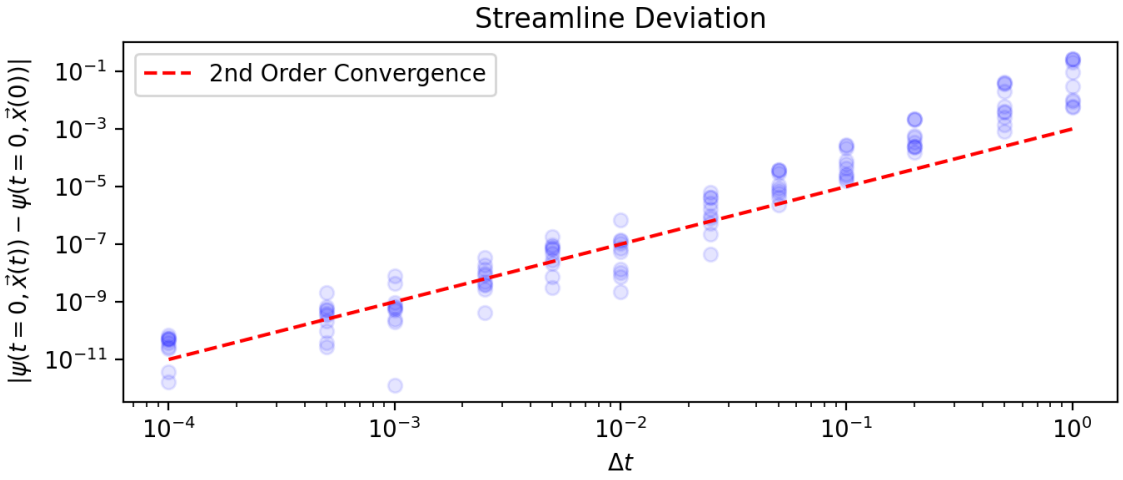
4.8. **Verification of Unperturbed Taylor-Green Flow.** As was stated previously, we initialize our Voriticity-Streamfunction solver as perturbed Taylor-Green flow. However, since Taylor-Green flow has an analytical solution, if we remove the perturbation, disable artificial viscosity forcing, and track the difference in solutions, we can verify if our solver is working correctly.



It is interesting to note that even when there is no perturbation, numerical errors cause some level of 'micro-turbulence' on the order of 10^{-12}. This figure can be reproduced by running two instances of `TG_verfication_run.py` (runtime < 5 m) with `gamma` set to 0.00 and 0.02, respectively. Afterwards, `TG_verification_plot.py` can be run (runtime < 25 s).

4.9. **Verification of Non-Inertial Particle Transport.** In order to test our implementation of non-inertial particle transport, we will similarly use the Taylor-Green analytical

solution. The positions for the streamlines, or lines with constant ψ are unchanging in time, meaning, that a fluid particle will always follow a closed loop. As a result, if we transport a particle, and compare its streamfunction at the end of some integration period to its initial streamfunction value, we can determine the 'drift' of the particle, or the Streamline Deviation.



It was shown that integrating up to time $t = 5$ and calculating the streamline deviation with decreasing Δt yields 3rd-Order convergence. This figure can be reproduced by running `non-inertial_verification.py` (runtime < 10 m).

4.10. **Verification of Inertial Particle Transport.** At the moment, no significant verification of inertial particle transport was completed.

5. Continued Research

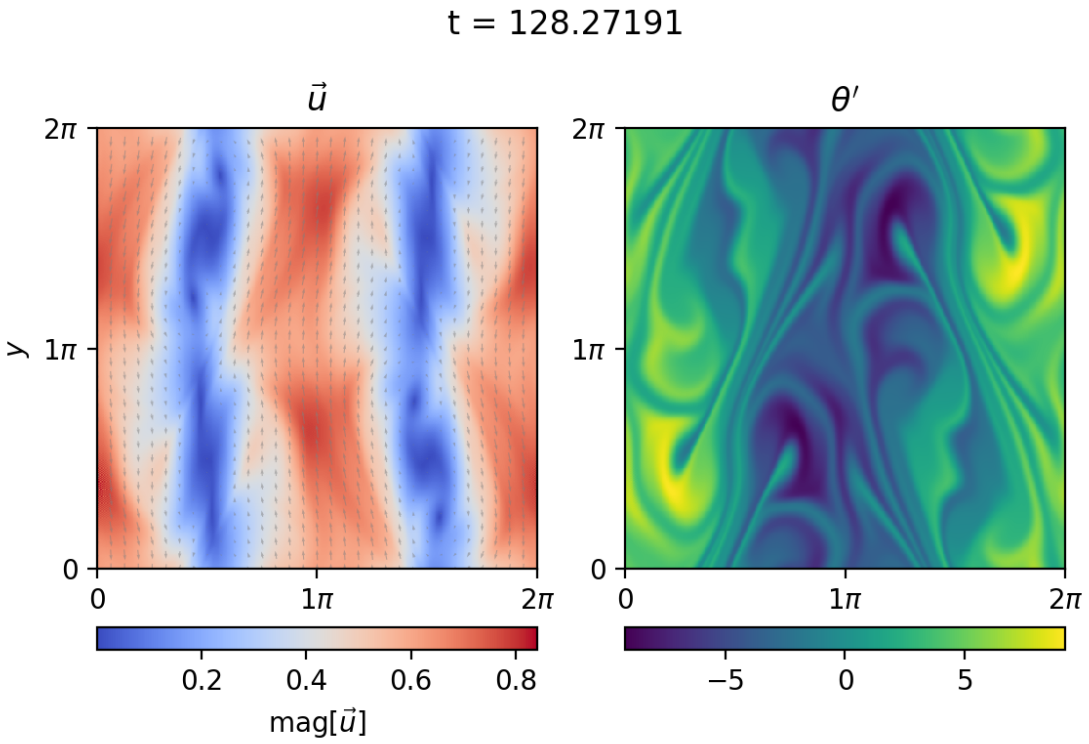
5.1. **Passive Scalar Transport.** The transport equation for a passive, diffusing scalar variable, i.e. temperature, is as follows:

$$\frac{\partial \theta}{\partial t} + u_x \frac{\partial \theta}{\partial x} + u_y \frac{\partial \theta}{\partial y} + f(u_x, u_y) = \frac{\nu}{c}\left(\frac{\partial^2 \theta}{\partial x^2} + \frac{\partial^2 \theta}{\partial y^2}\right) \tag{46}$$

where $f()$ is some arbitrary function that describes a constant scalar gradient, c is a dimensionless variable that is the ratio of scalar diffusion to momentum diffusion (viscosity). For temperature, this is the Prandtl Number Pr, for mass, this is the Schmidt Number Sc, etc.

This was completed, though the code will not be shared (though one can perhaps dig through the entire folder if need be). With the skills developed by the reader in the process of reading and implementing what has been carefully described in this document, this scalar transport can be implemented relatively directly.

Note: Plotting $\theta' = \theta - f(u_x, u_y)$ often results in a much better looking plot.



5.2. **Binary Controlled Turbulent Kinetic Energy.** Without forcing, due to viscosity, the flow will naturally decay, even with perturbations. In order to sustain turbulence, an artificial viscosity can be implemented.

Our viscous integrating factor $\Xi(t) = \exp\left[\nu(k_p^2 + k_q^2)t\right]$. We will define a slightly different version, where $\Xi_f(t) = \exp[\nu k_\Xi t]$. k_Ξ is defined as:

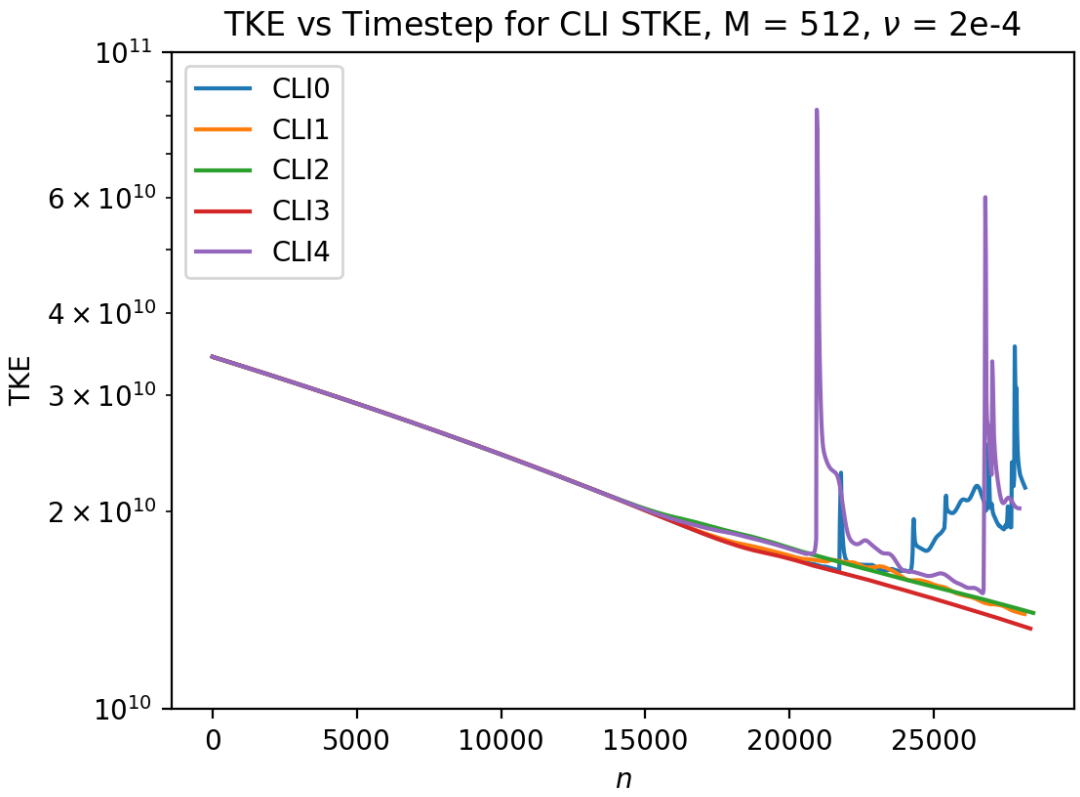
$$k_\Xi = \begin{cases} -k_p^2 - k_q^2, & 5^2 < k_p^2 + k_q^2 < 6^2 \\ 8(k_p^2 + k_q^2), & k_p^2 + k_q^2 < 4^2 \\ +k_p^2 + k_q^2, & \text{otherwise} \end{cases} \tag{47}$$

This is non-physical, as it is equivalent to having viscosity be negative for very fine structures, and boosted for very large structures. This can be used, but numerical instability issues are difficult to control. As a result, a binary switch was implemented that uses the standard $k_p^2 + k_q^2$ when the turbulent kinetic energy TKE is greater than the initial condition, and k_Ξ otherwise. The turbulent kinetic energy can be found as follows[15]:

$$\text{TKE} = \sum_i \sum_j \left[u_{x,ij}^2 + u_{y,ij}^2\right] = \sum_p \sum_q \left[|U_{x,pq}|^2 + |U_{y,pq}|^2\right] \tag{48}$$

However, even this seems plagued with odd instabilities.

[15]The sum in physical and frequency space can be interchanged via Parceval's Thereom



These instabilities were unfortunately not resolved, but they were well outside the scope of the initial goals of this project. Perhaps the reader may be able to resolve it.

6. Conclusion

Overall, this project was extremely helpful in helping me to develop skills in implementing a complex fluid dynamics method with almost no prior experience. I would like to thank Dr. Matheou for all his help during this project, as well as NIUVT for sponsoring this undergraduate research. I hope that this documentation serves to help the next researcher that will follow me.