5-2020

Applying the Generalized Quasilinear Approximation to Taylor-Couette Flow

Morgan Baxter
Bates College, mbaxter@bates.edu

Applying the Generalized Quasilinear Approximation to Taylor-Couette Flow

Morgan Baxter

Department of Physics and Astronomy, Bates College, Lewiston, ME 04240

Applying the Generalized Quasilinear Approximation to Taylor-Couette Flow

An Honors Thesis

Presented to the Department of Physics and Astronomy

Bates College

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Morgan Baxter

Lewiston, Maine

April 1, 2020

Contents

Acknowledgments

First and foremost, thanks to my parents for supporting me and encouraging me to challenge myself in whatever I do. Thanks to Avi Brach-Neufeld for his script for plotting linear instability growth rates. Editing and clarity help was provided by my grandmother, Professor Nancy Baxter Hastings. Special thanks to my adviser, Professor Jeffery Oishi, who provided the original 3D Taylor-Couette flow script, as well as wrote the GQL projection operator. I greatly appreciate your guidance, insight, and patience through this year-long process. Finally, thank you to all of my mathematics and physics teachers, going back to middle school, who instilled in me a passion for problem-solving and an interest in the world around me. I cannot express how lucky I am to have been taught and inspired by so many passionate teachers.

Morgan Baxter
3/31/2020

CHAPTER 1

Introduction to Fluid Dynamics

1. What is a fluid?

Fluids are a type of matter that deforms when shear force is applied to it. The study of fluids that are in motion is known as fluid dynamics, and it has a variety of applications in different areas of physics, including stellar mechanics, atmospheric science, and aerodynamics. Fluid dynamics is all around us, from the blood circulating in our bodies to the aerodynamics of our cars, from water flowing in rivers, pipes, and ocean currents, to atmospheric turbulence that affects our weather. As a result, being able to characterize and accurately predict these flows has been a topic of intense research in physics. The most popular methods used for research in this field involve computational fluid dynamics, or CFD. We seek to better understand ways to model chaotic fluid motion, better known as turbulence, and to develop techniques to decrease simulation costs.

Modeling fluids presents significant challenges because bulk fluids are made up of atoms numbering over the order of magnitude of Avogadro's number (10^{23}), so directly calculating the motion and interactions of every atom is not feasible. Instead, researchers use state variables, such as pressure or velocity, in a field that comprises the volume being studied. These state variables have scalar or vector values at every point in the field, and are iterated using the Navier-Stokes equations that govern fluid motion. There are a variety of ways to discretize the Navier-Stokes equations, including the finite element and finite volume techniques, as well as spectral methods, which we use.

No matter the method used for solving them, the incompressible Navier-Stokes equations remain the same. They are often written as a pair, referred to as the momentum (1.1) and continuity (1.2) equations.

$$\partial_t \boldsymbol{u} + (\boldsymbol{u} \cdot \boldsymbol{\nabla})\boldsymbol{u} = -\frac{\boldsymbol{\nabla} p}{\rho} + \nu \nabla^2 \boldsymbol{u} \tag{1.1}$$

$$\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0 \tag{1.2}$$

These represent a set of nonlinear, second order partial differential equations. Here, the variables are the velocity field $\boldsymbol{u}$, density ρ, kinematic viscosity ν, and pressure p. These equations are derived from conservation laws for momentum, mass, and energy, and the Cauchy momentum equation. The nonlinearity of this system stems from the term on the left hand side of the first equation, $(\boldsymbol{u} \cdot \boldsymbol{\nabla})\boldsymbol{u}$. It is precisely this nonlinear term that makes these equations so difficult to solve. There are a variety of approaches that can be taken to address this, including our approach, known as spectral methods. Fluids exhibit a variety of phenomena that can be modeled and studied, but we seek to apply spectral methods to the task of researching the transition between laminar and turbulent flow.

2. Turbulent versus laminar flow

Turbulence is defined as a fluid state where the velocity or pressure fields are highly chaotic. It is often described in contrast with laminar flow, in which layers of fluid move past each other with little or no mixing. Predicting when fluids transition from laminar to turbulent flow is an unsolved problem in physics. For example, in a pipe with water flowing inside, at a certain flow velocity, the fluid will transition from laminar flow to a more turbulent state. The transition from laminar to turbulent flow is marked by the development of eddies, vortices, and other chaotic disturbances at all scales. In aerodynamics and other areas of applied fluid dynamics, turbulence often results in undesired effects such as increased drag or decreased lift. As a result, engineering projects that incorporate a fluid regime are often designed to slow or even prevent the onset of turbulence. Therefore, the study of turbulent fluid motion is of great importance to a variety of physics and engineering disciplines.

A dimensionless quantity known as the Reynolds number is useful in determining transitions to turbulence. It is defined as the ratio between kinetic energy and viscous damping, and higher Reynolds are associated with turbulent behaviors. We study Taylor-Couette flow at various Reynolds numbers to investigate the phenomena of turbulence onset. We further nondimensionalize the Navier-Stokes equations to simplify calculations; this is discussed in chapter 2.

3. Investigation goals

Our primary goal for this investigation is to find under what circumstances the use of the generalized quasilinear (GQL) approximation is valid. Furthermore, we want to investigate whether the GQL approximation provides better cost improvement for systems with non-normal operators than systems without them. There has already been research on the transient growth of perturbations in the Taylor-Couette problem due to linear nonnormal mechanisms, where laminar flows "become turbulent due to finite amplitude perturbations which are transiently amplified by nonnormal mechanisms." [1] We seek to show that GQL is useful for continuing these investigations into nonnormal operators.

We utilize the Dedalus framework to solve our system of differential equations as an initial value problem, or IVP. Dedalus allows for utilizing parallel computing as well as already-existing powerful tools for solving our differential equations. Dedalus's powerful analysis toolset allows us to compare flow characteristics such as kinetic energy and average flow velocity, in order to determine the accuracy of the GQL approximation. We carry out one DNS and multiple GQL runs (at different Λ values) and compare results in the form of snapshots of flow velocity fields, average flow velocity, and kinetic energy.

CHAPTER 2

Methods

1. Taylor-Couette Flow

Taylor-Couette flow is a very well-studied experimental setup in the area of fluid stability research. A version of the setup was described by Maurice Couette around 1890, for the purpose of measuring viscosity. It was then used by Geoffrey Taylor in 1923 to investigate phenomena of hydrodynamic stability, and confirmed the no-slip condition, where a viscous fluid has zero velocity at the boundary with a solid surface. Taylor-Couette flow is of interest because there is not only a considerable amount of research on fluid behaviors in the experiment,[2, 3, 4, 5, 6] but also because it presents a fluid setup that is periodic in two of three dimensions. This greatly simplifies the experimental setup, and allows us to ignore the effects of other surfaces interacting with the fluid.

The reasoning behind choosing Taylor-Couette flow to study fluid instability is that turbulence is exceptionally hard to characterize. Laminar flow is very ordered and stable, where turbulence is disordered and chaotic. Here, we use chaotic in the mathematical sense: chaotic systems are those that, from small differences in starting parameters, evolve very differently from each other. Taylor-Couette flow has been extensively researched, and the various turbulent phenomena that occur at different inner and outer rotation speeds have been well-characterized, as demonstrated in Figure 2.1. The simplicity of the geometry and parametric setup of the experiment means we can replicate the literature and verify our script replicates previous experiments before invoking the GQL approximation. The regions that we investigate in this figure are marked with purple stars. They both lie on the axis where the outer cylinder rotation is zero, and our Reynolds numbers are 750, 100, and 1500, which lie in the wavy vortex flow, the boundary with modulated wavy vortices, and Turbulent Taylor vortices zones, respectively.

The experimental setup is made up of two concentric cylinders of different diameters as seen in Figure 2.2. Fluid fills the space between the two cylinders, and the cylinders can be considered to be infinite in height. This is to negate the effects of the cylinder cap on the fluid behavior. Here, R_1 and R_2 are the cylinder radii, and Ω_1 and Ω_2 are rotation rates.

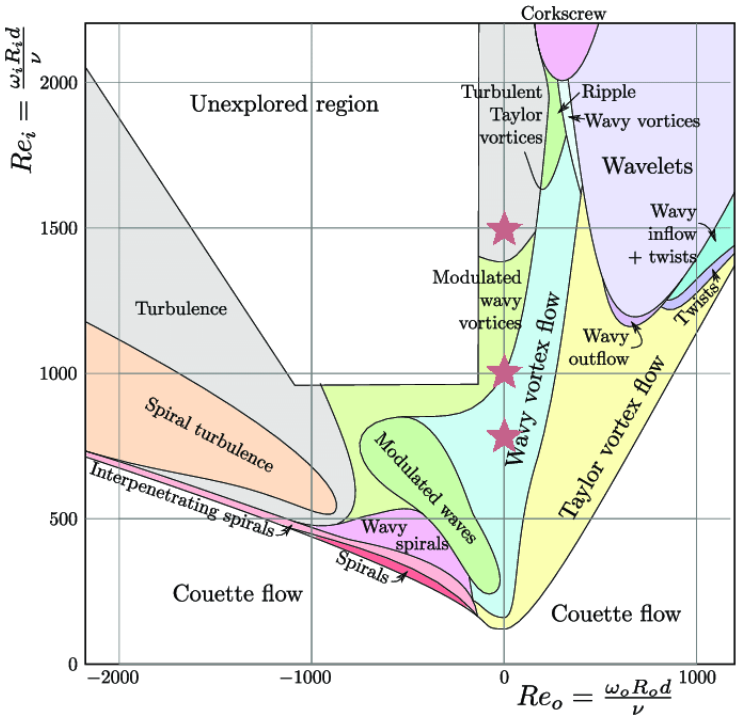


FIGURE 2.1. Reproduced from [7] with permission

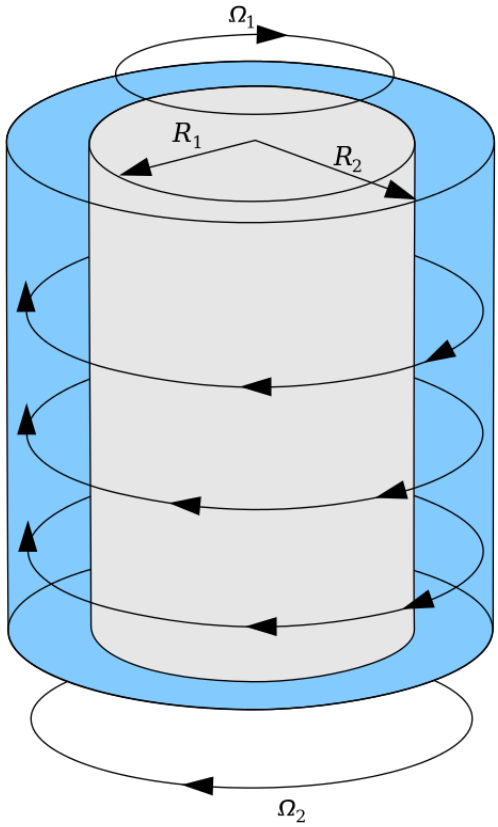


FIGURE 2.2. Experimental setup of Taylor-Couette Flow, from [8] and used with permission

As can be seen from Figure 2.2, it makes the most sense to use a cylindrical coordinate system (r, θ, z) for our numerical calculations. It is worth noting that the order of the axes is switched to (z, θ, r) due to software constraints in Dedalus. These three bases correspond to (u, v, w) as the scalar coefficients of the velocity fields in $(\boldsymbol{z}, \boldsymbol{\theta}, \boldsymbol{r})$ directions.

2. Introduction to spectral methods

There are a variety of methods that can be employed in computational fluid dynamics. For example, various finite-difference methods are effective when calculating flow around objects with complex geometries, such as supersonic airflow over a wing, or compressed through a turbine. We will be utilizing spectral methods, which yield optimal resolution per computational element. Rather than store values for velocity at every grid point in a volume, spectral methods decompose these state variables into sums of basis functions. Then, the coefficients for each basis function are iterated forward through time.

Much of the following is reproduced from a paper describing the Dedalus framework [9]. Rather than storing flow variables such as pressure and velocity at discrete points, areas, or volumes, these state variables are truncated series of polynomials, which Dedalus provides as pre-defined sets. Each term in the expansion is a mode, so using 32 modes in the θ direction, for example, means the flow in the θ direction is represented by a polynomial series truncated at 32 modes. No matter what polynomial basis we use, we use N_c to denote the number of modes we are using. Higher numbers of modes offer higher resolution, but can also increase cost. Each basis has a different native interval, and Dedalus includes methods for each basis to produce a set of collocation points N_g, defined by the native grid of scale s, where $s = N_g/N_c$. The Dedalus framework which we use for running our simulations has support for Fourier, Chebyshev, Legendre, Hermite, Laguerre, and compound bases. For our three dimensions, we use two bases, Fourier and Chebyshev. The utility of these two bases becomes clear when we consider their roots. Setting $i = 0, 1, ..., N_g - 1$, we can examine the roots of both the Fourier and Chebyshev bases. The Fourier roots are Equation 2.1 and the Chebyshev roots are Equation 2.2.

$$x_i = \frac{2\pi i}{N_g} \tag{2.1}$$

$$x_i = -\cos\left(\frac{\pi(i + 1/2)}{N_g}\right) \tag{2.2}$$

The choice of these two bases is intimately tied to our choice of geometry. As a result of the no-slip condition, fluid in contact with the cylinder walls must move at the same velocity as the wall. This represents a forced boundary condition for our system. We chose to use a Chebyshev basis for the r direction partially in order to resolve the details of fluid behavior close to the cylinder walls, and also because the behavior is non periodic, unlike in the θ and z directions. The Fourier basis is evenly spaced across $[0, 2\pi]$, in contrast with the Chebyshev basis, where near 0, there is an even distribution of points, versus near the end points $[-1, 1]$, where points are considerably denser, allowing for the resolution of smaller structures. The Chebyshev basis offers an even distribution near the center of the collocation grid, where $\Delta x \approx \pi/N_g$, but near the edges, grid point density increases quadratically. This is useful in the r direction, where the center of the flow between the cylinders doesn't demand as high of a resolution as the boundary layers near the edges. The flow is periodic in both θ and z, and therefore we use Fourier polynomials for these basis. It is easy to see why it makes sense to have periodicity

in θ, while the periodicity in z is consistantly observed in Taylor-Couette experimental setups. Figure 2.3 below provides a good visualization of the collocation points for the two bases:

FIGURE 2.3. Example collocation grids for Fourier and Chebyshev bases. Reproduced from [9] with permission.

Our Fourier bases can be written as the complex exponential modes, on the native interval $[0, 2\pi]$. Here, the superscript F indicates that this is a Fourier series.

$$\phi_k^F(x) = e^{ikx} \tag{2.3}$$

Next, we define a function $f(x)$ that serves as the coefficients for the modes. This is a sum over all positive and negative wavenumbers. Here, $k_m = \text{floor}((N_c - 1)/2)$ is the maximum resolved wavenumber.

$$f(x) = \sum_{-k_m}^{k_m} f_k \phi_k^F(x) \tag{2.4}$$

We then solve for expansion coefficients using the fast Fourier transform (FFT), which are given explicitly by:

$$f_k = \frac{1}{2\pi} \int_0^{2\pi} f(x)\phi_k^{F*}(x)dx = \frac{1}{N_g} \sum_{i=0}^{N_g-1} f(x_i)\phi_k^{F*}(x_i) \tag{2.5}$$

For our Chebyshev basis, we use the following on the native interval $[-1, 1]$:

$$T_n(x) = \cos(n \cos^{-1}(x)) \tag{2.6}$$

A function $f(x)$ is defined as:

$$f(x) = \sum_{n=0}^{N_c-1} f_n T_n(x) \tag{2.7}$$

We can then solve for the coefficients f_n:

$$f_n = \frac{2 - \delta_{n,0}}{\pi} \int_{-1}^{1} \frac{f(x)T_n(x)}{\sqrt{1 - x^2}} dx = \frac{2 - \delta_{n,0}}{N_g} \sum_{i=0}^{N_g-1} f(x_i)T_n(x_i) \tag{2.8}$$

Further reading about spectral methods and Dedalus solver implementation can be found in the Dedalus methods paper. [9]

3. Why Polynomials?

When dealing with problems that require solving differential equations, it is clear that making our basis functions easy to differentiate can make solving the DEs significantly more straightforward. Therefore, we use sets of polynomials that are easy to differentiate, such as the complex exponential Fourier series. Not only are these polynomials easy to differentiate, they represent orthogonal basis sets with respect to the modes we want to model. Polynomial orthogonality

can be illustrated with the following example. If we take two functions $f(x)$ and $g(x)$, each composed of two modes (mode 1 and mode 2), we can see how the concept of projection works.

$$f(x) = ae^{ikx} + be^{i2kx} \tag{2.9}$$

$$g(x) = ce^{ikx} + de^{i2kx} \tag{2.10}$$

When we multiply these two functions together, projections onto other modes become apparent.

$$f(x)g(x) = \left(ae^{-ikx} + be^{i3kx}\right)\left(ce^{-i2kx} + de^{i5kx}\right) = ace^{-i3kx} + bce^{ikx} + bde^{i8kx} + ade^{i4kx} \tag{2.11}$$

In this way, the multiplication of two functions with first and second modes results into projections onto the second, third, and fourth modes. This is trivial when dealing with very few modes, but the number of operations increases considerably when dealing with 64, 128, or even 256 modes. Therefore, we are interested in ways to decrease the cost involved. The quasilinear approximation provides considerable speed increases, but at the cost of precision. We propose that the general quasilinear approximation offers a flexible compromise between resolving small-scale phenomena with increased speed.

4. The Generalized Quasilinear Approximation

The study of fluid dynamics is significantly complicated by the wide range of scales involved. Turbulent systems exhibit behaviors from the macroscopic to the molecular level, from the generation of vortices at a visible scale to thermal dissipation at the smallest scales. As a result, modeling turbulent systems often requires decisions to be made about the smallest scales to include in the computation. Direct numerical simultation, or DNS, is the simulation of all scales of fluid behavior without any approximation. The result is highly accurate at all scales, but can be very expensive. As a result, there is considerable interest by fluid researchers of all backgrounds to find approximations that preserve dynamics of systems at all scales while also decreasing computation cost. The generalized quasilinear approximation, or GQL, was first proposed by Marston, Chini, and Tobias in 2016, with the goal of significantly decreasing the processing power required to simulate fluid behaviors by selectively allowing phenomena at differing scales to interact. [10]

GQL is a form of direct statistical simulation (DSS) that allows for lower-cost, high resolution spectral analysis. We choose an integer Λ that delineates low and high modes, where $|m| \leq \Lambda$ is a low mode, where $|m| > \Lambda$ is a high mode. The GQL retains all nonlinear interactions in the low modes, allowing the mean flow to demonstrate all DNS behavior. However, any high modes that result from combinations of low modes are discarded, for the purpose of keeping the simulation inexpensive. The result is that energy is distributed at small scales (high modes) through interactions with the fully nonlinear mean flow (low modes). This is the key improvement of GQL over the quasilinear approximation, since the presence and effects of eddies and smaller-scale phenomena are preserved and have similar results to DNS. Not only does GQL have the advantage that low modes are allowed to undergo all nonlinear interactions, but through nonlocal spectral transfer of high modes through interaction with large scales, we keep interactions that would be discarded in the quasilinear case. There have been a series of investigations into GQL, many of which have involved using Dedalus. [11, 12]

Considering interactions between modes, there are a total of six permutations that are possible. These are illustrated in Figure 2.4. High modes that combine to produce low modes, low modes that combine to produce low modes, and low and high interactions that produce high

modes are kept. All other interactions are omitted. In the case of the Navier-Stokes momentum equation, the GQL can be seen as:

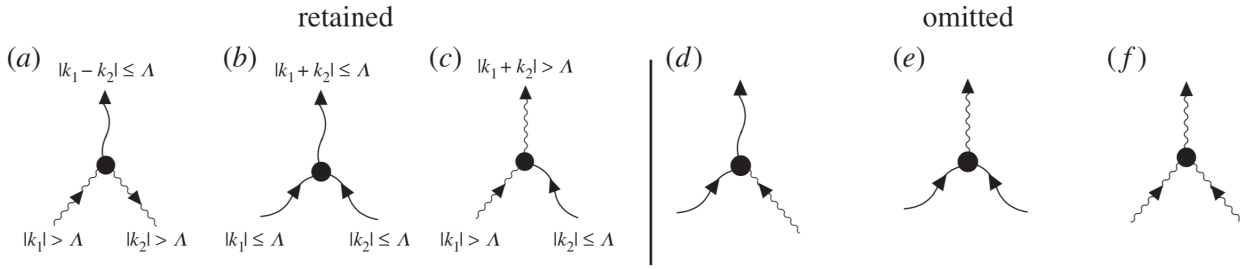


FIGURE 2.4. Feynmann diagrams demonstrating how GQL keeps and discards modes. Reproduced from [12] with permission.

We can rewrite the velocity field $\boldsymbol{u}$ as $\boldsymbol{u} = \boldsymbol{u}_h + \boldsymbol{u}_l$. We denote high modes using the subscript $_h$ and low modes with the subscript $_l$. Our nonlinear term from Equation 1.1 $(\boldsymbol{u}\cdot\boldsymbol{\nabla})\boldsymbol{u}$ then can be written as:

$$(\boldsymbol{u}\cdot\boldsymbol{\nabla})\boldsymbol{u} = ((\boldsymbol{u}_h+\boldsymbol{u}_l)\cdot\boldsymbol{\nabla})(\boldsymbol{u}_h+\boldsymbol{u}_l) = (\boldsymbol{u}_h\cdot\boldsymbol{\nabla})\boldsymbol{u}_h + (\boldsymbol{u}_l\cdot\boldsymbol{\nabla})\boldsymbol{u}_h + (\boldsymbol{u}_h\cdot\boldsymbol{\nabla})\boldsymbol{u}_l + (\boldsymbol{u}_l\cdot\boldsymbol{\nabla})\boldsymbol{u}_l \tag{2.12}$$

We can further expand each term above using the definition of the material derivative in cylindrical coordinates. For example, the interactions between a high and low mode can be expanded as follows:

$$\begin{aligned}(\boldsymbol{u}_h\cdot\boldsymbol{\nabla})\,\boldsymbol{u}_l = &\left(u_h\frac{\partial u_l}{\partial r} + \frac{v_h}{r}\frac{\partial u_l}{\partial\theta} + w_h\frac{\partial u_l}{\partial z} + \frac{v_h v_l}{r}\right)\hat{\boldsymbol{r}}\\ &+\left(u_h\frac{\partial v_l}{\partial r} + \frac{v_h}{r}\frac{\partial v_l}{\partial\theta} + w_h\frac{\partial v_l}{\partial z} + \frac{v_h u_l}{r}\right)\hat{\boldsymbol{\theta}}\\ &+\left(u_h\frac{\partial w_l}{\partial r} + \frac{v_h}{r}\frac{\partial w_l}{\partial\theta} + w_h\frac{\partial w_l}{\partial z}\right)\hat{\boldsymbol{z}}\end{aligned} \tag{2.13}$$

From Equation 1.1, we can write our momentum equation for the GQL approximation:

$$\partial_t\boldsymbol{u} + \langle(\boldsymbol{u}_l\cdot\boldsymbol{\nabla})\boldsymbol{u}_h + (\boldsymbol{u}_h\cdot\boldsymbol{\nabla})\boldsymbol{u}_l\rangle_h + \langle(\boldsymbol{u}_l\cdot\boldsymbol{\nabla})\boldsymbol{u}_l + (\boldsymbol{u}_h\cdot\boldsymbol{\nabla})\boldsymbol{u}_h\rangle_l = -\frac{\boldsymbol{\nabla}p}{\rho} + \nu\nabla^2\boldsymbol{u} \tag{2.14}$$

Here, we use $\langle\rangle_l$ and $\langle\rangle_h$ to represent projection onto the low and high modes, respectively. The projection operator takes the result of the interactions and discards the modes that do not project onto either the high or low modes. For the equation for the low modes, we are interested in the combinations of low and low, and high and high modes that project onto the low modes, or $\langle(\boldsymbol{u}_l\cdot\boldsymbol{\nabla})\boldsymbol{u}_l + (\boldsymbol{u}_h\cdot\boldsymbol{\nabla})\boldsymbol{u}_h\rangle_l$, and for the high modes, we want combinations of either low and high or high and low modes: $\langle(\boldsymbol{u}_l\cdot\boldsymbol{\nabla})\boldsymbol{u}_h + (\boldsymbol{u}_h\cdot\boldsymbol{\nabla})\boldsymbol{u}_l\rangle_h$. Therefore, Equation 2.14 represents the version of Equation 1.1 with the GQL approximation implemented.

There have been a variety of investigations into applying GQL to various fluid geometries, including barotropic jets, rotating Couette flow, the helical magnetorotational instability, and flows in a thermal annulus.[10, 13, 11, 12] We seek to apply GQL to a full 3D Taylor-Couette flow and search for values of Λ that allow for the simulation of nonlinear dynamics at a considerably lower cost than DNS.

5. Parameters and Initial Conditions

For our Taylor-Couette system, four parameters can define the whole experimental setup: the radius of the first and second cylinders, R_1 and R_2, and their rotation rates, Ω_1 and Ω_2. We choose units for time, distance, and velocity in order to considerably simplify our governing equations. We set our length unit to be the gap width, which we enforce to be equal to be $\delta = 1$. Our time unit is $\frac{\delta}{R_1\Omega_1} = \frac{1}{R_1\Omega_1}$, and our velocity unit is $R_1\Omega_1$. We also define an aspect ratio Γ that is the ratio between the height and the width of the cylinders. This allows us to calculate the 'height' of the cylinders in the $\hat{z}$ direction: $L_z = \frac{\Gamma}{R_2 - R_1}$. In order to normalize the gap between the cylinders, we define η as the ratio between R_1 and R_2. As a result, we can express the cylinder radii as:

$$R_1 = \frac{\eta}{1-\eta}, R_2 = \frac{1}{1-\eta} \tag{2.15}$$

We define the Reynolds number to be $Re = \frac{R_1\Omega_1}{\nu}$. Higher Reynolds numbers tend to predict the onset of turbulence, and lower Reynolds numbers predict laminar flow. We also keep the outer cylinder still while the inner cylinder rotates at $\Omega_1 = \frac{1}{R_1}$. We define $\Omega_2 = \mu\Omega_1$, but set $\mu = 0$, setting the outer cylinder rotation rate equal to zero. This results in our parameters for each run being our Reynolds number Re, radii ratio η, and aspect ratio Γ. For GQL runs, we also specify Λ values for both z and θ, but for the purposes of this investigation, we set them to be equal.

For each of our three coordinates r, θ, z, we write the velocity component in that direction, u, v, w. Also, in order to decrease the number of derivatives of the velocity component in a dimension with respect to r, we also define ur, vr, wr, which represent the partial derivative of the velocity field with respect to r. The full script used to run the simulations can be found in Appendix B.

For all our runs, we used initial conditions from Willis's PhD thesis. [14] These produced pertubations in the u and w fields. The variable $m1$ is the parameter that gives the wavenumber of the initial transverse wave travelling in the $\hat{\theta}$ direction. For the following equations, we define new variables x and $k + z$ as follows: $x = r - R_1$, $k_z = \frac{2\pi}{L_z}$.

$$u = k_z^2 x^2 (1-x)^2 \sin k_z z 10^{-3} \tag{2.16}$$

$$w = \frac{k_z x^2 (1-x)^2 \cos(k_z z)}{r} + 2k_z \cos(k_z x)((1-x)^2 x - x^2(1-x^2)) - \frac{(x^2(1-x)^2)}{r} m_1 \cos(m_1\theta) 10^{-3} \tag{2.17}$$

When studying turbulent flows, it is often useful to subtract the mean flow from the perturbations in order to study them better. Laminar flow in a Taylor-Couette system is called Couette flow, and it is axisymmetric and not a function of z, so it is only a function or r. Our cylinder raddii ratio is η, and μ represents the ratio between the rotation rates: $\mu = \frac{\Omega_2}{\Omega_1}$. For some of our analysis, we use this definition of Couette flow to subtract it from the entire flow, so that we may better analyze the perturbations. We define Couette flow as v_0 below, along with variables A and B to simplify writing the expression:

$$A = \frac{(\eta^{-1} - 1)(\mu - \eta^2)}{1 - \eta^2} \tag{2.18}$$

$$B = \frac{\eta(1-\mu)}{(1-\eta)(1-\eta^2)} \tag{2.19}$$

$$v_0(r) = Ar + \frac{B}{r} \tag{2.20}$$

6. Carrying out simulations

Carrying out high-resolution simulations of our experimental setup in three dimensions is too computationally demanding to run on a single computer in a reasonable timeframe. In order to significantly speed up the process of acquiring data, we run our script in parallel on Leavitt, a computing cluster at Bates College. Leavitt is made up of 16 nodes, each with two processors with 14 cores, for a total of 448 cores. We use Open MPI, an open source software package that parallelizes by dividing the problem so it can be solved across multiple cores.

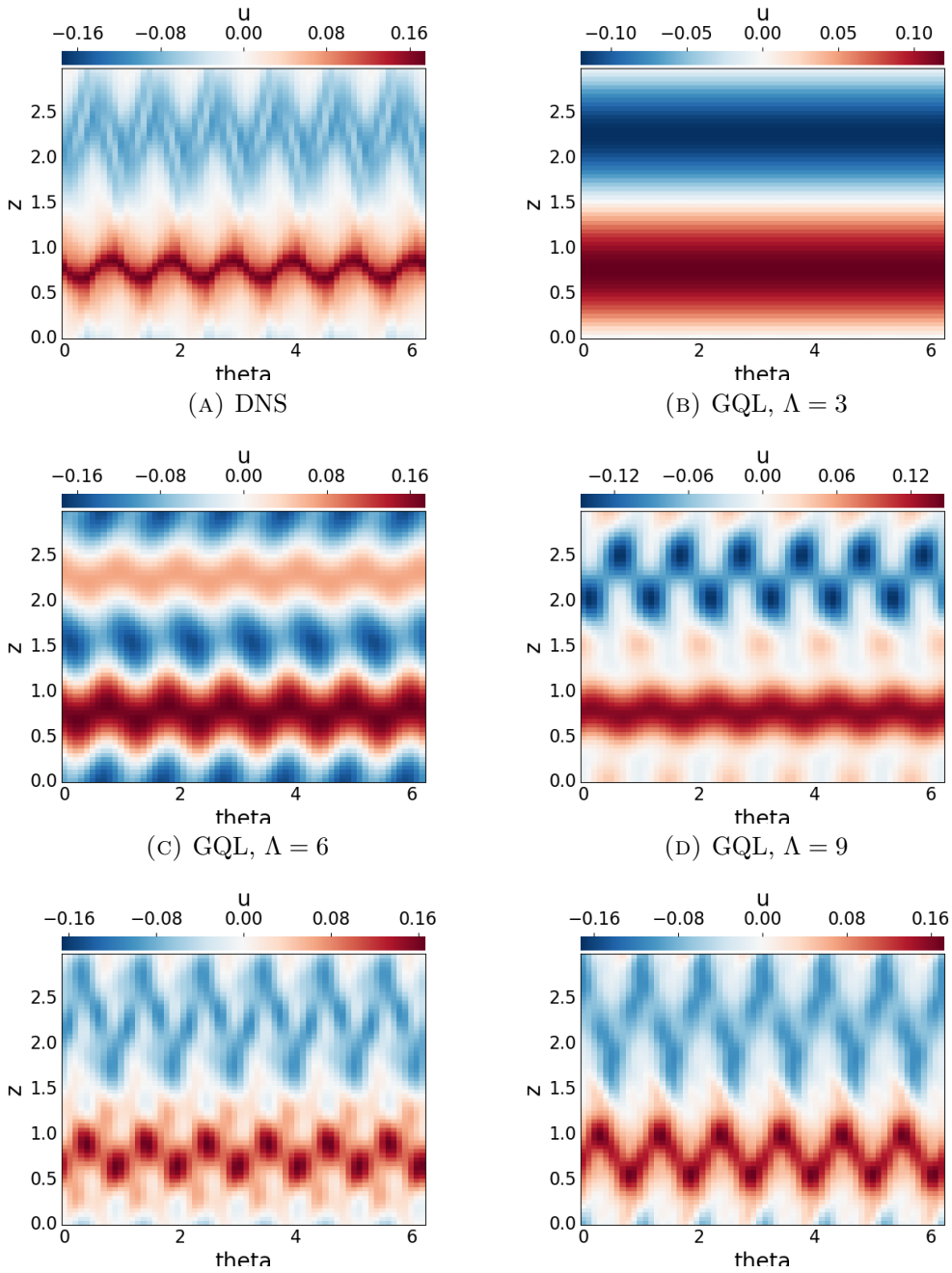
As discussed in Section 2, we use Fourier modes for the z and θ directions, and Chebyshev in r. Because we need to solve equations simultaneously across r, we cannot distribute r modes across processors, so we parallelize over modes in z and θ, creating "pencils" in r. First, consider the number of modes in our 3D problem. If we run with 64 modes in each dimension, the total number of modes is 64^3! Previous research has found that the optimal number of pencils per core is close to 8, so with a total of $64^2 = 4096$ pencils divided by 64 cores, we have 64 pencils per core. GQL is slower on 128 cores at 32 pencils per core, which we don't understand at the moment.

There are a series of analysis tasks that are written into the script that allow for data such as angular momentum or average flow velocity to be written out at each timestep. Beyond just looking at the movement of fluid in our system, we use momentum, kinetic energy, and plane and volume average analysis tasks to get quantitative data about the flow. These come in the form of slices, two dimensional datasets, radial data, one-dimensional datasets, and scalars, such as kinetic energy. Our implementation of these analysis tasks can be seen in our code, included in Appendix B.

CHAPTER 3

Results

Research on fluids often involves a variety of analysis tools, the first of which is often just examining the flow velocity fields. Below are a series of snapshots, or views of the flow at a fixed time. We can see that the small-scale features in the DNS run are entirely smoothed out at lower Λ, but start to reappear as we increase the Λ values. The $m = 6$ wavenumber is clearly visible in the DNS, $\Lambda = 15$, and $\Lambda = 12$ runs, but is less resolved in $\Lambda = 9$, and $\Lambda = 6$ runs. The run with the lowest Λ value demonstrates that as a result of so many modal interactions being discarded, we reach a steady solution with no traveling wave in the θ direction. Since Taylor-Couette is ultimately a statistical problem, snapshots of the flow such as these are not as useful as other quantitative measures of the flow. However, these snapshots are an effective method of illustrating the way that the GQL approximation affects the flow characteristics at different Λ values. The GQL runs here begin to resemble DNS as we increase the Λ value, which was expected, and at lower values, it deviates to much smoother solutions.



(A) DNS

(B) GQL, $\Lambda = 3$

(C) GQL, $\Lambda = 6$

(D) GQL, $\Lambda = 9$

(E) GQL, $\Lambda = 12$

(F) GQL, $\Lambda = 15$

FIGURE 3.1. u, slicing through r at midpoint between cylinders, $Re = 1000$

Figures 3.2 and 3.3 show the changes in the kinetic energy of perturbations over time of different runs. Note that the y-axis is a log scale, and that the GQL runs grow at the same rate as the DNS run during the linear growth phase. This is a good consistency check that shows that the GQL method is working as expected, because the linear terms are all the same between DNS and GQL. We see some deviation from DNS when we're close to the saturation point, particularly for $\Lambda = 3$, where the energy deviates briefly from the other runs before converging to the same value. Note that the linear instability growth rate is higher for the $Re = 1500$ run

than the $Re = 750$ run. This is because at higher Reynolds numbers, the onset of turbulence occurs more quickly than at lower values.

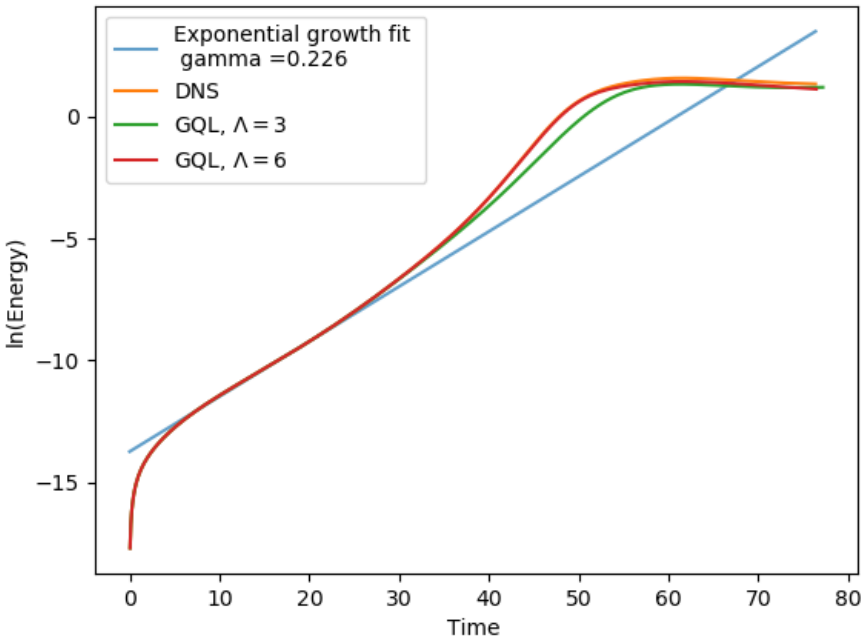


FIGURE 3.2. Kinetic energy versus time for DNS and GQL runs at $Re = 750$

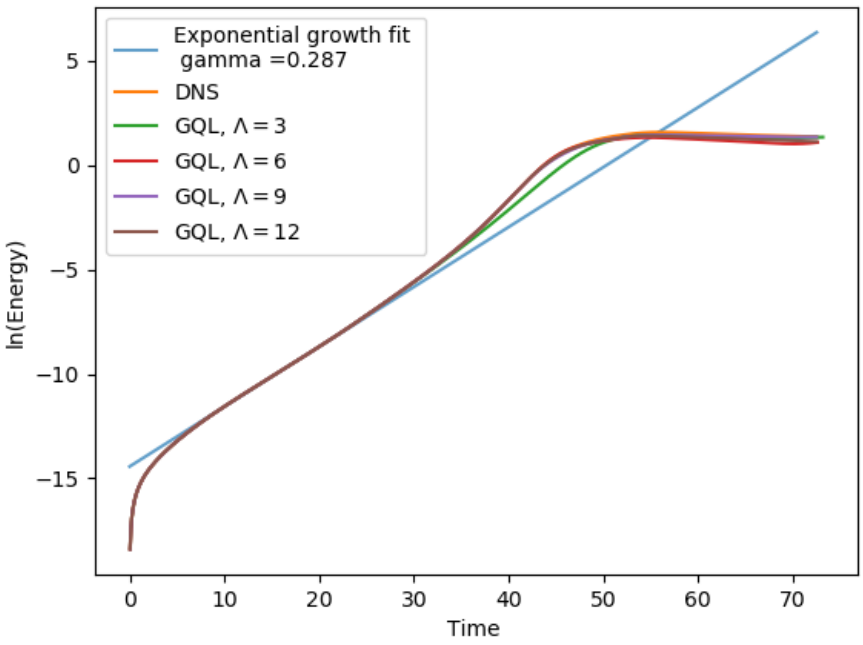


FIGURE 3.3. Kinetic energy versus time for DNS and GQL runs at $Re = 1500$

Figures 3.4, 3.5, and 3.6 plot the mean flow in the θ direction as a function of r. These figures illustrate the no-slip condition, as all the runs converge to velocity 1 at the inner cylinder and 0 at the outer cylinder. For all three Reynolds numbers, we can see that the GQL runs seem to predict a higher velocity in the middle of the gap, which is unexpected. The DNS curve is considerably smoother than the GQL curves, and the $\Lambda = 3$ run is the least smooth for all three, owing to the number of discarded modes. Despite the minor deviation from DNS, the GQL runs here behave as expected, and are within reasonable deviation. These plots also illustrate the effect of increasing the Reynolds number, as the higher values of Re result in a quicker transition from the no-slip boundary condition to steady average flow in the center of the channel. These three plots illustrate the average at the last time slice, but averaging over the last 100 slices would likely result in GQL solutions that match the DNS run.

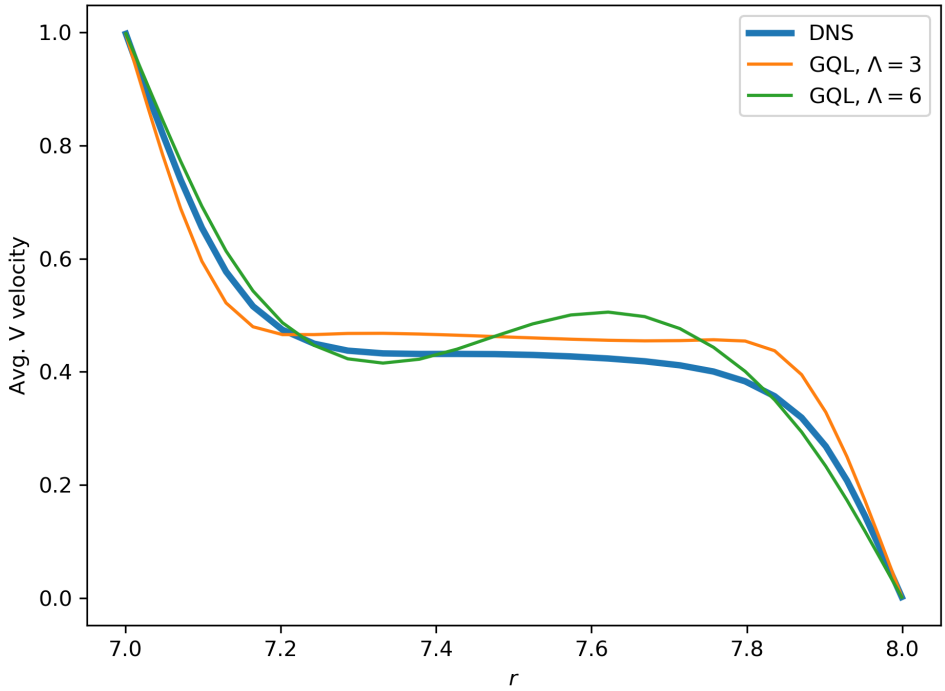


FIGURE 3.4. v_{total} averaged over z and θ as a function of r, $Re = 750$

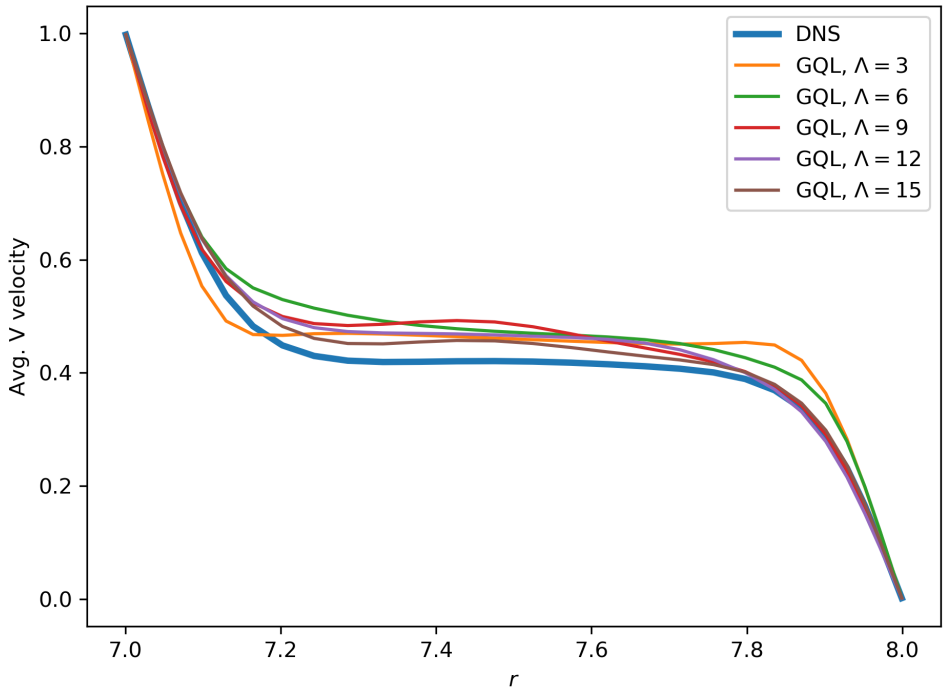


FIGURE 3.5. v_{total} averaged over z and θ as a function of r, $Re = 1000$

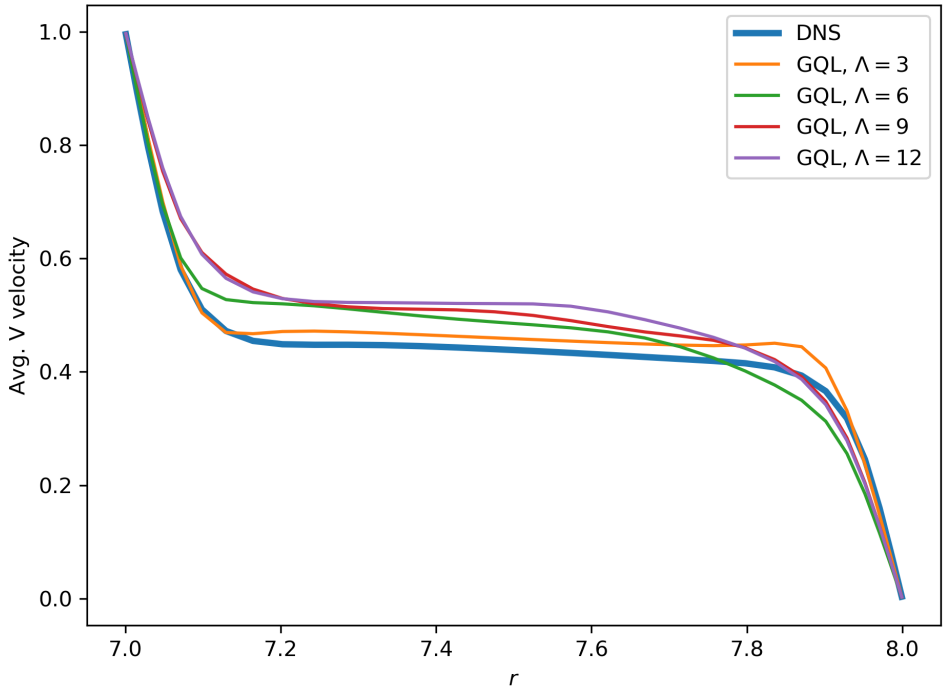
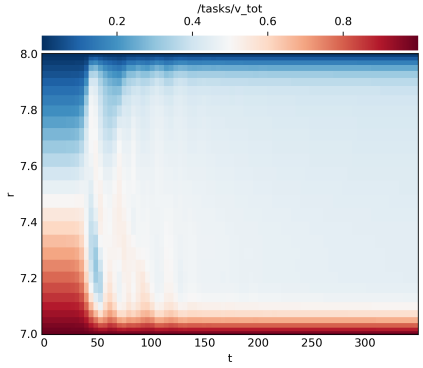
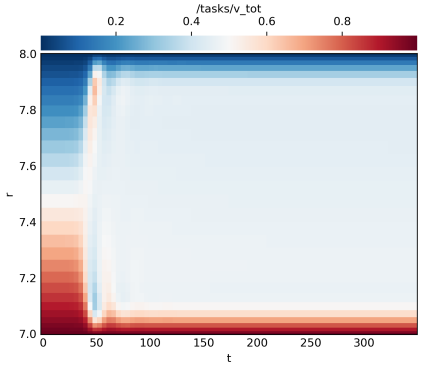


FIGURE 3.6. v_{total} averaged over z and θ as a function of r, $Re = 1500$

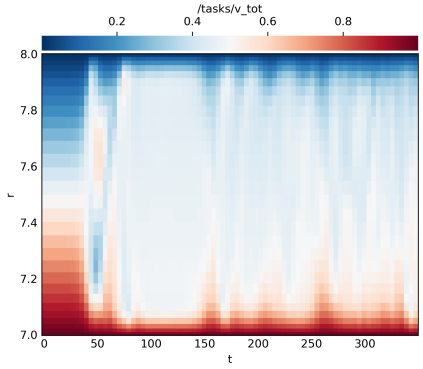
Figure 3.7 plots the average velocity in the $\hat{\theta}$ direction against time for different values of Λ. The initial gradient subsides into an average flow except near the cylinder surfaces. What is noteworthy about these plots is that the $\Lambda = 3$ GQL reaches and maintains the same steady state solution as DNS, where the higher Λ value runs experience bursts of velocity that are not captured by either DNS or $\Lambda = 3$. The stripe at $t = 0$ is a linear gradient, representing Couette flow. These plots also illustrate the no-slip condition, as the stripes at $r = 7$ and $r = 8$ are constant for all the runs, as the cylinder rotation rates are constant. Note that the transition from the initial Couette flow to the steady state solution occurs at the same time for all the runs, another good consistency check between DNS and GQL. These results are consistent with Figure 3.6, as the DNS and $\Lambda = 3$ runs have lower average v_{total} than the higher Λ value runs. As with Figures 3.4, 3.5, and 3.6, averaging the GQL runs in time could result in better agreement with the DNS steady state result.



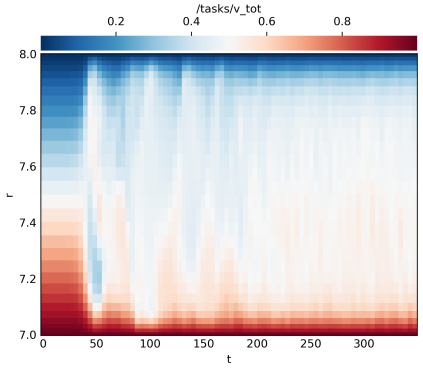
(A) DNS



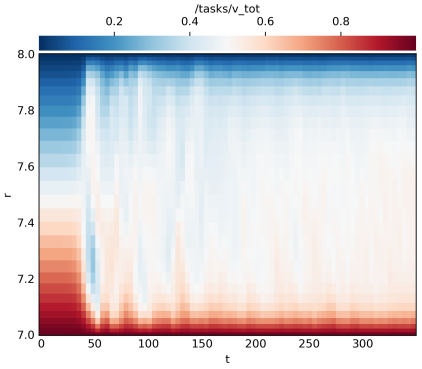
(B) GQL, $\Lambda = 3$



(C) GQL, $\Lambda = 6$



(D) GQL, $\Lambda = 9$



(E) GQL, $\Lambda = 12$

FIGURE 3.7. v_{total}, slicing through r at midpoint between cylinders, $Re = 1500$

CHAPTER 4

Discussion and Applications

Researchers utilizing techniques of resolvent analysis may find that GQL can offer conisderably better performance over DNS techniques. GQL approximations could be used by researchers in aerodynamics utilizing spectral methods to simulate nonlinear behavior. [15] Furthermore, a variety of astrophysical systems are uniquely suited to take advantage of the GQL approximation for complex magnetohydrodynamic simulations. Any spectral model that exhibits nonlinear behavior could potentially take advantage of this approximation.

Our implementation of the GQL approximation to three-dimensional Taylor-Couette flows demonstrates that GQL can be a useful tool for spectral analysis of nonlinear flows. We note a deviation in kinetic energy for the GQL runs from DNS, but the deviation is minor and likely due to a coding error. We expect that at high Λ values, we should get very good agreement for kinetic energy, but this doesn't appear to be the case. However, we expect to find this error and have the GQL runs converge to the same kinetic energy value as DNS. Furthermore, a rewrite of our GQL operator could yield performance increases by only carrying out the nonlinear interactions that will be retained, rather than carrying out the full nonlinear calculations and omitting modes according to GQL and our chosen Λ value.

Further investigation using these tools could involve modal analysis of growth rates and utilizing different geometries and experimental setups. In the future, the GQL approximation can be used to tell us about the most important nonlinear terms in a prototypical nonlinear flow.

Bibliography

[1] A. Meseguer. Non-normal effects in the taylorcouette problem. *Theoretical and Computational Fluid Dynamics*, 16:71–77, 01 2002.

[2] K. T. Coughlin and P. S. Marcus. Modulated waves in Taylor-Couette flow. Part 1. Analysis. *Journal of Fluid Mechanics*, 234:1–18, January 1992.

[3] K. T. Coughlin and P. S. Marcus. Modulated waves in Taylor-Couette flow. Part 2. Numerical simulation. *Journal of Fluid Mechanics*, 234:19–46, January 1992.

[4] Jeremy Goodman and Hantao Ji. Magnetorotational instability of dissipative Couette flow. *Journal of Fluid Mechanics*, 462(1):365–382, Jul 2002.

[5] Michael Krygier and Roman Grigoriev. Numerical investigation of direct laminar-turbulent transition in counter-rotating Taylor-Couette flow. In *APS Division of Fluid Dynamics Meeting Abstracts*, APS Meeting Abstracts, page M6.003, November 2015.

[6] Tommy Dessup, Laurette S. Tuckerman, Jos Eduardo Wesfreid, Dwight Barkley, and Ashley P. Willis. Self-sustaining process in taylor-couette flow. *Physical Review Fluids*, 3(12), Dec 2018.

[7] Daniel Duda, Marek Klimko, Radek Skach, Jan Uher, and Vaclav Uruba. Hydrodynamic education with rheoscopic fluid. *EPJ Web of Conferences*, 213:02014, 01 2019.

[8] Wikipedia, the free encyclopedia. File:couettetaylorsystem.svg, 2011. [Online; accessed April 1, 2020].

[9] Keaton J. Burns, Geoffrey M. Vasil, Jeffrey S. Oishi, Daniel Lecoanet, and Benjamin P. Brown. Dedalus: A Flexible Framework for Numerical Simulations with Spectral Methods. *Physical Review Research, submitted*, page arXiv:1905.10388, May 2019.

[10] J. B. Marston, G. P. Chini, and S. M. Tobias. Generalized Quasilinear Approximation: Application to Zonal Jets. *Physical Review Letters*, 116(21):214501, May 2016.

[11] S. M. Tobias, J. S. Oishi, and J. B. Marston. Generalized quasilinear approximation of the interaction of convection and mean flows in a thermal annulus. *Proceedings of the Royal Society of London Series A*, 474(2219):20180422, Nov 2018.

[12] S. M. Tobias and J. B. Marston. Three-dimensional rotating Couette flow via the generalised quasilinear approximation. *Journal of Fluid Mechanics*, 810:412–428, January 2017.

[13] S. E. Clark and J. S. Oishi. The Weakly Nonlinear Magnetorotational Instability in a Global, Cylindrical Taylor-Couette Flow. *The Astrophysical Journal*, 841:2, May 2017.

[14] Ashley Phillip Willis. The hydromagnetic stability of taylor-couette flow, 2002.

[15] Shenren Xu. Simplified Eigenvalue Analysis for Turbomachinery Aerodynamics with Cyclic Symmetry. *arXiv e-prints*, page arXiv:1912.08460, December 2019.

Appendix A

```
import numpy as np
from dedalus.core.field import Operand
from dedalus.core.operators import Operator, FutureField, Interpolate

class GQLProjection(Operator, FutureField):
    """
    Projection operator for generalized quasilinear approximation
    """
    def __init__(self, arg, cutoff, subspace, dim=None,**kw):
        arg = Operand.cast(arg)
        super().__init__(arg,**kw)
        self.cutoff = cutoff

        # by default, execute GQL on all but the last dimension
        if not dim:
            self.dim = self.domain.dim - 1
        else:
            self.dim = dim

        local_coeff = self.domain.all_elements()
        low_mask = np.ones(self.domain.local_coeff_shape, dtype='bool')

        for i in range(self.dim):
            low_mask &= (np.abs(local_coeff[i]) <= cutoff[i])
        if subspace == 'high' or subspace == 'h':
            self.mask = ~low_mask
            subspace_name = 'h'
        elif subspace == 'low' or subspace == 'l':
            self.mask = low_mask
            subspace_name = 'l'
        else:
            raise ValueError("Subspace must be high/h or low/l, not
            ↪  {}".format(subspace))

        cutoff_str = ",".join([str(i) for i in cutoff])
        self.name = 'Proj[({}),{}]'.format(cutoff_str,subspace_name)

    def meta_constant(self, axis):
```

```
    # Preserve constancy
    return self.args[0].meta[axis]['constant']

def check_conditions(self):
    """Projection must be in coefficient space"""
    return self.args[0].layout is self._coeff_layout

def operate(self, out):
    for i in range(self.dim):
        self.args[0].require_layout('c')
    out.data[:] = self.args[0].data
    out.data *= self.mask
```

Appendix B

```
"""
Usage:
  taylor_couette_3d.py --re=<reynolds> --eta=<eta> --m=<initial_m>
↪  [--ar=<Gamma>] [--restart=<restart>] [--restart_file=<restart_file>]
↪  --mesh_1=<mesh_1> --mesh_2=<mesh_2> [--GQL=<GQL>] [--Lambda_z=<Lambda_z>]
↪  [--Lambda_theta=<Lambda_theta>]
  taylor_couette_3d.py

Options:
  --re=<reynolds>  Reynolds number for simulation
  --eta=<eta>      Eta - ratio of R1/R2
  --m=<initial_m>   M1 mode to begin initial conditions
  --ar=<Gamma>     Aspect ratio (height/width)
  --mesh1=<mesh_1> First mesh core count
  --mesh2=<mesh_2> Second mesh core count
  --restart=<restart> True or False
  --restart_file=<restart_file> Point to a merged snapshots_s1.h5 file
  --GQL=<GQL> True or False
  --Lambda_z=<Lambda_z> Specify an integer cutoff to seperate low and high
↪  modes for z
  --Lambda_theta=<Lambda_theta> Specify an integer cutoff to seperate low and
↪  high modes for theta
"""

import numpy as np
import h5py
from dedalus.extras import flow_tools
from dedalus import public as de
import time
import logging
from docopt import docopt
import os
import subprocess
from mpi4py import MPI
from GQLProjection import GQLProjection

de.operators.parseables["Project"] = GQLProjection
```

```
comm=MPI.COMM_WORLD
rank=comm.Get_rank()

args=docopt(__doc__)
Re1=float(args['--re'])
eta=np.float(args['--eta'])
Gamma = int(args['--ar'])
GQL = args['--GQL']

if GQL!=None:
    GQL=True
    Lambda_z = int(args['--Lambda_z'])
    Lambda_theta = int(args['--Lambda_theta'])
mesh_1 = int(args['--mesh_1'])
mesh_2 = int(args['--mesh_2'])
m1 = int(args['--m'])
restart = bool(args['--restart'])

root = logging.root
for h in root.handlers:
    h.setLevel("INFO")

logger = logging.getLogger(__name__)

if restart==True:
    restart_file = str(args['--restart_file'])
"""
delta = R2 - R1
mu = Omega2/Omega1
eta = R1/R2

scale [L] = delta
scale [T] = delta/(R1 Omega1)
scale [V] = R1 Omega1

Default parameters from Barenghi (1991, J. Comp. Phys.).
"""

mu = 0
#eta = 0.8770
# ~ Re1 = 80
#Lz = 2.0074074463832545
Sc = 1
dealias = 3/2
nz=64
ntheta=64
nr=32
```

```
eta_string = "{:.4e}".format(eta).replace(".","-")


if GQL:
    root_folder =
    ↪  "TC_3d_re_{}_eta_{}_Gamma_{}_M1_{}_{}_{}_{}_GQL_Lambdaz_{}_Lambdat_{}/".format(Re
    ↪  Lambda_theta)
else:
    root_folder =
    ↪  "TC_3d_re_{}_eta_{}_Gamma_{}_M1_{}_{}_{}_{}/".format(Re1,eta_string,Gamma,m1,nz,n

path = 'results/'+root_folder
if rank==0:
    if not os.path.exists(path):
        os.mkdir(path)
    elif restart==False:
        logger.info('Folder for run already exists.')
        logger.info('Use restart, rename existing folder, or change
        ↪  parameters')
        #subprocess.call(['analysis_scripts/./kill_script.sh'])
sim_name=path
restart=False

#derived parameters
R1 = eta/(1. - eta)
R2 = 1./(1-eta)
Omega1 = 1/R1
Omega2 = mu*Omega1
nu = R1 * Omega1/Re1
midpoint = R1 + (R2-R1) / 2
Lz = Gamma * (R2 - R1)

#Taylor Number
Ta = ((1+eta)**4/(64*eta**2) ) * ( (R2-R1)**2 * (R2+R1)**2 *
↪  (Omega1-Omega2)**2 ) / nu**2 #Grossman lohse
Ta1 = (2*Omega1**2*(R2-R1)**4*eta**2 ) / (nu**2 *(11-eta**2)  )
Ro_inv = (2 * Omega2 * (R2-R1) ) /  (np.abs(Omega1-Omega2)*R1 )


logger.info("Re:{:.3e}, eta:{:.4e}".format(Re1,eta))
logger.info("Taylor Number:{:.2e}, Ro^(-1):{:.2e}".format(Ta,Ro_inv))

logger.info("Lz set to {:.6e}".format(Lz))

variables = ['u','ur','v','vr','w','wr','p']
```

```
#domain
z_basis = de.Fourier(  'z', nz, interval=[0., Lz], dealias=dealias)
theta_basis = de.Fourier('theta', ntheta, interval=[0., 2*np.pi],
↪  dealias=dealias)
r_basis = de.Chebyshev('r', nr, interval=[R1, R2], dealias=3/2)

bases = [z_basis, theta_basis, r_basis]
# ~ bases = t_bases + r_basis
domain = de.Domain(bases, grid_dtype=np.float64, mesh=[mesh_1,mesh_2])

#problem
problem = de.IVP(domain, variables=variables)

#params into equations
problem.parameters['eta']=eta
problem.parameters['mu']=mu
problem.parameters['Lz']=Lz
problem.parameters['nu']=nu
problem.parameters['R1']=R1
problem.parameters['R2']=R2
problem.parameters['pi']=np.pi

#Substitutions

"""
this implements the cylindrical del operators.
NB: ASSUMES THE EQUATION SET IS PREMULTIPLIED BY A POWER OF r (SEE BELOW)!!!

Lap_s --> scalar laplacian
Lap_r --> r component of vector laplacian
Lap_t --> theta component of vector laplacian
Lap_z --> z component of vector laplacian

"""

problem.substitutions['A'] = '(1/eta - 1.)*(mu-eta**2)/(1-eta**2)'
problem.substitutions['B'] = 'eta*(1-mu)/((1-eta)*(1-eta**2))'


problem.substitutions['v0'] = 'A*r + B/r'       #background profile? forcing
↪  instead of forcing the boundaries
problem.substitutions['dv0dr'] = 'A - B/(r*r)'  #d/dr of background forcing

problem.substitutions['v_tot'] = 'v0 + v'       #total velocity in v
↪  direction. (azimuthal)
```

```
problem.substitutions['vel_sum_sq'] = 'u**2 + v_tot**2 + w**2'
problem.substitutions['plane_avg_r(A)'] = 'integ(integ(r*A,
↪  "z"),"theta")/(2*pi*r*Lz)'
#problem.substitutions['plane_avg_z(A)'] = 'integ(integ(A, "r"),"theta")/Lz'
problem.substitutions['vol_avg(A)']   = 'integ(r*A)/(pi*(R2**2 - R1**2)*Lz)'
problem.substitutions['probe(A)'] = 'interp(A,r={}, theta={}, z={})'.format(R1
↪  + 0.5, 0., Lz/2.)

problem.substitutions['KE'] = '0.5*vel_sum_sq'
problem.substitutions['u_rms'] = 'sqrt(u*u)'
problem.substitutions['v_rms'] = 'sqrt(v*v)'
problem.substitutions['w_rms'] = 'sqrt(w*w)'
problem.substitutions['Re_rms'] = 'sqrt(vel_sum_sq)*Lz/nu'
problem.substitutions['epicyclic_freq_sq']  = 'dr(r*r*v*v)/(r*r*r)'

problem.substitutions['enstrophy'] = '0.5*((dtheta(w)/r - dz(v_tot))**2 +
↪  (dz(u) - wr )**2 + (vr + dv0dr + v_tot/r - dtheta(u))**2)'

 # not pre-multiplied...don't use this in an equation!
problem.substitutions['DivU'] = "ur + u/r + dtheta(v)/r + dz(w)"


# assume pre-multiplication by r*r
problem.substitutions['Lap_s(f, f_r)'] = "r*r*dr(f_r) + r*f_r +
↪  dtheta(dtheta(f)) + r*r*dz(dz(f))"
problem.substitutions['Lap_r'] = "Lap_s(u, ur) - u - 2*dtheta(v)"
problem.substitutions['Lap_t'] = "Lap_s(v, vr) - v + 2*dtheta(u)"
problem.substitutions['Lap_z'] = "Lap_s(w, wr)"

# substitutions
if GQL:

    # substitutions for projecting onto the low and high modes
    problem.substitutions['Project_high(A)'] =
    ↪  "Project(A,[{:d},{:d}],'h')".format(Lambda_z, Lambda_theta)
    problem.substitutions['Project_low(A)'] =
    ↪  "Project(A,[{:d},{:d}],'l')".format(Lambda_z, Lambda_theta)

    problem.substitutions['u_l'] = "Project_low(u)"
    problem.substitutions['u_h'] = "Project_high(u)"
    problem.substitutions['v_l'] = "Project_low(v)"
    problem.substitutions['v_h'] = "Project_high(v)"
    problem.substitutions['w_l'] = "Project_low(w)"
    problem.substitutions['w_h'] = "Project_high(w)"

    # r momentum (GQL)
```

```
    problem.add_equation("r*r*dt(u) - nu*Lap_r - 2*r*v0*v + r*v0*dtheta(u) +
    ↪  r*r*dr(p) = r*v0*v0 - Project_high(r*r*u_h*dr(u_l)+ r*v_h*dtheta(u_l)
    ↪  + r*r*w_h*dz(u_l) - r*v_h*v_l + r*r*u_l*dr(u_h) +r*v_l*dtheta(u_h) +
    ↪  r*r*w_l*dz(u_h) - r*v_h*v_l) - Project_low(r*r*u_h*dr(u_h)+
    ↪  r*v_h*dtheta(u_h) + r*r*w_h*dz(u_h) - r*v_h*v_h + r*r*u_l*dr(u_l) +
    ↪  r*v_l*dtheta(u_l) + r*r*w_l*dz(u_l) - r*v_l*v_l)")

    # theta momentum (GQL)
    problem.add_equation("r*r*dt(v) - nu*Lap_t + r*r*dv0dr*u + r*v0*u +
    ↪  r*v0*dtheta(v) + r*dtheta(p) = - Project_high(r*r*u_h*dr(v_l) +
    ↪  r*v_h*dtheta(v_l) + r*r*w_h*dz(v_l) + r*v_h*u_l + r*r*u_l*dr(v_h) +
    ↪  r*v_l*dtheta(v_h) + r*r*w_l*dz(v_h) + r*v_l*u_h) -
    ↪  Project_low(r*r*u_h*dr(v_h) + r*v_h*dtheta(v_h) + r*r*w_h*dz(v_h) +
    ↪  r*v_h*u_h + r*r*u_l*dr(v_l) + r*v_l*dtheta(v_l) + r*r*w_l*dz(v_l) +
    ↪  r*v_l*u_l)")

    # z momentum (GQL)
    problem.add_equation("r*r*dt(w) - nu*Lap_z + r*r*dz(p) + r*v0*dtheta(w) =
    ↪  - Project_high(r*r*u_h*dr(w_l) + r*v_h*dtheta(w_l) + r*r*w_h*dz(w_l) +
    ↪  r*r*u_l*dr(w_h) + r*v_l*dtheta(w_h) + r*r*w_l*dz(w_h)) -
    ↪  Project_low(r*r*u_h*dr(w_h) + r*v_h*dtheta(w_h) + r*r*w_h*dz(w_h) +
    ↪  r*r*u_l*dr(w_l) + r*v_l*dtheta(w_l) + r*r*w_l*dz(w_l))")

else:

    # DNS nonlinear terms
    problem.substitutions['UdotGrad_s(f, f_r)'] = "r*r*u*f_r + r*v*dtheta(f) +
    ↪  r*r*w*dz(f)"
    problem.substitutions['UdotGrad_r'] = "UdotGrad_s(u, ur) - r*v*v"
    problem.substitutions['UdotGrad_t'] = "UdotGrad_s(v, vr) + r*u*v"
    problem.substitutions['UdotGrad_z'] = "UdotGrad_s(w, wr)"

    # momentum equations
    problem.add_equation("r*r*dt(u) - nu*Lap_r - 2*r*v0*v + r*v0*dtheta(u) +
    ↪  r*r*dr(p) = r*v0*v0 - UdotGrad_r")
    problem.add_equation("r*r*dt(v) - nu*Lap_t + r*r*dv0dr*u + r*v0*u +
    ↪  r*v0*dtheta(v) + r*dtheta(p)  = -UdotGrad_t  ")
    problem.add_equation("r*r*dt(w) - nu*Lap_z + r*r*dz(p) + r*v0*dtheta(w) =
    ↪  -UdotGrad_z")

#continuity
problem.add_equation("r*ur + u + dtheta(v) + r*dz(w) = 0")

#Auxillilary equations
problem.add_equation("ur - dr(u) = 0")
problem.add_equation("vr - dr(v) = 0")
```

```
problem.add_equation("wr - dr(w) = 0")


#Boundary Conditions
problem.add_bc("left(u) = 0")
problem.add_bc("right(u) = 0", condition="ntheta != 0 or nz != 0")
problem.add_bc("right(p) = 0", condition="ntheta == 0 and nz == 0")
problem.add_bc("left(v) = 0")
problem.add_bc("right(v) = 0")
problem.add_bc("left(w) = 0")
problem.add_bc("right(w) = 0")



#Create solver
solver = problem.build_solver(de.timesteppers.RK443)


#intial conditions
u = solver.state['u']
ur = solver.state['ur']
v = solver.state['v']
vr = solver.state['vr']
w = solver.state['w']
wr = solver.state['wr']
r = problem.domain.grid(-1,scales=problem.domain.dealias)
z = problem.domain.grid(0,scales=problem.domain.dealias)
theta = problem.domain.grid(1,scales=problem.domain.dealias)
r_in = R1

willis=True

if restart==True:
        logger.info("Restarting from file {}".format(restart_file))
        write, last_dt = solver.load_state(restart_file, -1)
elif willis==True:
    ## Willis & Bahrenghi ICs
    logger.info("Using initial conditions from Willis's PhD thesis")

    A0 = 1e-3
    #m1=3
    u.set_scales(domain.dealias, keep_data=False)
    w.set_scales(domain.dealias, keep_data=False)
    x = r - r_in
    kz = 2*np.pi/Lz
    logger.info('kz : {}'.format(kz))
    u['g'] = A0 * kz**2 * x**2 * (1-x)**2 * np.sin(kz*z)
```

```
    w['g'] = A0 *( kz * x**2 * (1-x)**2 * np.cos(kz*z)/r + 2*kz*np.cos(kz*z) *
    ↪  ((1-x)**2 * x - x**2 * (1 - x)) - (x**2 * (1 - x)**2)/r * m1 *
    ↪  np.cos(m1*theta))
    u.differentiate('r',out=ur)
    w.differentiate('r',out=wr)
else:
    # Random perturbations to v in (r, z)
    A0 = 1e-3
    logger.info("Using axisymmetric noise initial conditions in v with
    ↪  amplitude A0 = {}.".format(A0))
    v.set_scales(domain.dealias, keep_data=False)
    gshape = domain.dist.grid_layout.global_shape(scales=domain.dealias)
    slices = domain.dist.grid_layout.slices(scales=domain.dealias)
    rand = np.random.RandomState(seed=42)
    noise = rand.standard_normal(gshape)
    slices_axi = [slices[0], 0, slices[-1]]
    noise = noise[slices_axi][:, None, :] *
    ↪  np.ones(np.shape(noise[slices_axi]))
    v['g'] = A0 * noise[slices] * np.sin(np.pi * (r - r_in))
    v.differentiate('r', out=vr)


#Setting Simulation Runtime
omega1 = 1/eta - 1.
period = 2*np.pi/omega1
solver.stop_sim_time = 8*period
solver.stop_wall_time = 24*3600.#np.inf
solver.stop_iteration = np.inf

#CFL stuff
CFL = flow_tools.CFL(solver, initial_dt=1e-2, cadence=5,
↪  safety=0.3,max_change=1.5, min_change=0.5,max_dt=1)
CFL.add_velocities(('u', 'v', 'w'))

# Flow properties
flow = flow_tools.GlobalFlowProperty(solver, cadence=10)
flow.add_property("abs(DivU)", name='divu')
flow.add_property("integ(r*KE)", name='KE')
flow.add_property("integ(r*enstrophy)", name='enstrophy')

dt = CFL.compute_dt()
# Main loop


geo_factor = 1
```

```
#Analysis

output_time_cadence = 0.1*period
scalar_output_time_cadence = output_time_cadence/100.

# ~ analysis =
↪  solver.evaluator.add_file_handler('taylor_couette',scalar_output_time_cadence,max_w
logger.info("sim_name= {}".format(sim_name))
snapshots=solver.evaluator.add_file_handler(sim_name +
↪  'snapshots',sim_dt=output_time_cadence,max_writes=np.inf)
snapshots.add_system(solver.state)

#Analysis files
Jeffs_analysis=True
if Jeffs_analysis:
    analysis_slice = solver.evaluator.add_file_handler(sim_name+"/slices",
    ↪  parallel=False, sim_dt=output_time_cadence)
    analysis_slice.add_task("interp(u,r={})".format(midpoint),
    ↪  name="u_slice",scales=4)
    analysis_slice.add_task("interp(v,r={})".format(midpoint),
    ↪  name="v_slice",scales=4)
    analysis_slice.add_task("interp(w,r={})".format(midpoint),
    ↪  name="w_slice",scales=4)

    analysis_slice.add_task("interp(KE, z=0)", name="KE")
    analysis_slice.add_task("plane_avg_r(v_tot)", name="v_tot")
    analysis_slice.add_task("plane_avg_r(u_rms)", name="u_rms")
    analysis_slice.add_task("plane_avg_r(v_rms)", name="v_rms")
    analysis_slice.add_task("plane_avg_r(w_rms)", name="w_rms")
    analysis_slice.add_task("plane_avg_r(Re_rms)", name="Re_rms")
    analysis_slice.add_task("plane_avg_r(epicyclic_freq_sq)",
    ↪  name="epicyclic_freq_sq")
    analysis_slice.add_task("integ(r*v, 'z')", name='Angular Momentum')

    analysis_profile = solver.evaluator.add_file_handler(sim_name+"/profiles",
    ↪  max_writes=20, parallel=False)


    analysis_spectra = solver.evaluator.add_file_handler(sim_name+"/spectra",
    ↪  max_writes=20, parallel=False)
    analysis_spectra.add_task("u", name="uc", layout="c")
    analysis_spectra.add_task("v", name="vc", layout="c")
    analysis_spectra.add_task("w", name="wc", layout="c")

    analysis_scalar = solver.evaluator.add_file_handler(sim_name+"/scalar",
    ↪  parallel=False,sim_dt=scalar_output_time_cadence)
```

```
    analysis_scalar.add_task("integ(r*KE)", name="KE")
    analysis_scalar.add_task("vol_avg(u_rms)", name="u_rms")
    analysis_scalar.add_task("vol_avg(v_rms)", name="v_rms")
    analysis_scalar.add_task("vol_avg(w_rms)", name="w_rms")
    analysis_scalar.add_task("vol_avg(Re_rms)", name="Re_rms")
    analysis_scalar.add_task("probe(w)", name="w_probe")
    analysis_scalar.add_task("integ(r*enstrophy)", name="enstrophy")
    analysis_scalar.add_task("integ(r*KE) - integ(v0, 'r')",
    ↪  name="pertubation_KE")



logger.info("Starting main loop...")
start_time = time.time()
while solver.ok:
    solver.step(dt)
    if (solver.iteration-1) % 100 == 0:
        logger.info('Iteration: %i, Time: %e, Inner rotation periods: %e, dt:
        ↪  %e' %(solver.iteration, solver.sim_time, solver.sim_time/period,
        ↪  dt))
        logger.info('Max |divu| = {}'.format(flow.max('divu')))
        logger.info('Total KE per Lz =
        ↪  {}'.format(geo_factor*flow.max('KE')/Lz))
        logger.info('Total enstrophy per Lz =
        ↪  {}'.format(geo_factor*flow.max('enstrophy')/Lz))
        #gc.collect()
    dt = CFL.compute_dt()

end_time = time.time()

logger.info('Simulation run time: {:f}'.format(end_time-start_time))

logger.info('Time per iteration:
↪  {:f}'.format((end_time-start_time)/solver.iteration))

logger.info('Simulation ended')
```