Computational Models using the Equatorial Beta Plane

A Senior 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

Michael Bancroft

Lewiston, Maine

April 2, 2018

Contents

List of Figures

Acknowledgments

Thank you to Dr. Jeffrey Oishi for his guidance throughout this research. He has taught me countless skills that I will be able to utilize in further research down the road.

Thank you to the entire physics department and all the professors that have helped me through my education here at Bates College. I came into Bates not knowing what I wanted to study and within days knew that the physics department was incredible and I have enjoyed working with everyone in the department.

Thank you to my family for all of their love and support. My parents have always pushed me and I hope this thesis can live up to their expectations. Also a special thank you to my brother and Fiance. They have always had my back and kept me moving forward.

CHAPTER 1

Abstract

The goal of this research was to study the problems that arise due to the mathematical formula used/required when studying Rossby waves in the Tachocline region of the sun and how they are affected by the Coriolis force. Error can arise because the formula for the Coriolis force has a sine term contained within it, which returns a zero when applied to behavior running parallel to the equator. In order to account for this we use the equatorial beta plane, which is an approximation for the Coriolis force. This research is attempting to use python framework Dedalus and Bates super computer Leavitt to create models of these waves. First we will model these waves at a latitude above the equator of the sun. Then we will see the affects when magneto-hydrodynamic turbulence is applied with varying strengths of magnetic fields.

CHAPTER 2

Introduction

The goal of this thesis is to explore if we can effectively use the periodic equatorial beta plane to create simulations of naturally occurring phenomena in the Tachocline region of the sun. In order to talk about the equatorial beta plane we must first examine the Coriolis force. This is a fictitious force that involves a curved path an object follows when traveling latitudinally across a rotating body. This force is given in the equation below where f is the Coriolis force, Ω is the angular velocity and ψ is the angle from the equator.

$$f = 2\Omega sin(\psi) \tag{2.1}$$

An issue arises with this equation when considering waves that propagate parallel to the equator and stay at or near zero latitude. This gives us a value for ψ of zero which outputs a value of zero for the Coriolis force. This issue can be navigated around by using the small angle approximation. This allows us to rewrite $sin(\psi)$ as roughly equal to y. From here we are able to use an alternate equation known as the equatorial beta plane.

This approximation allows us to maintain a Coriolis force greater than zero when considering waves parallel to the equator. This is achieved by using the latitudinal distance from the equator as the only variable. The equation for the equatorial beta plane can be found below where again; f is the Coriolis force, y is the latitudinal distance from the equator, and β is the beta plane constant $\frac{2\Omega}{a}$. Ω is the reference frame's angular rotation rate and a is its radius from the axis of rotation.

$$f = \beta y \tag{2.2}$$

The test case we are building this simulation for is based off the paper β-Plane Magneto-hydrodynamic Turbulence in the Solar Tachocline. In this paper they examine naturally occurring Rossby waves in the Tachocline region of the sun. This region (shown below) is important in driving solar magnetic activity and may also play a role in the mixing processes of the solar interior. Here we specifically want to simulate how magneto-hydrodynamic (mhd) turbulence and applied magnetic fields affect this region.

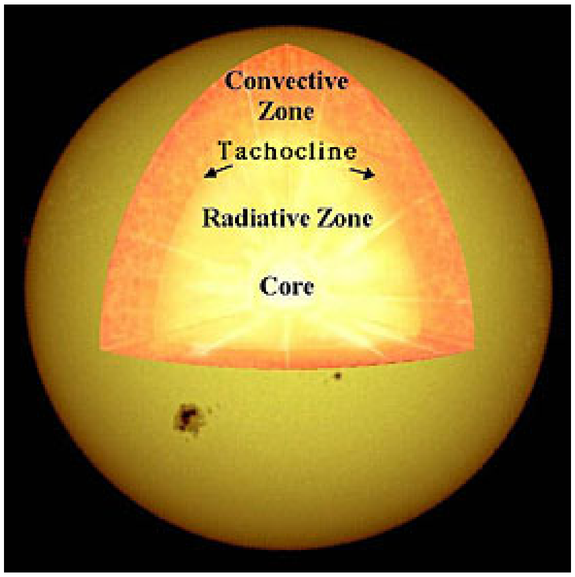


Figure 2.1. Model of the Sun showing the Tachocline region

The Tachocline is a transition region in the outer third of the sun between the radiative interior which exhibits solid body rotation and the differentially rotating convective zone which rotates as a normal fluid. One of the central questions surrounding this region is whether it is expanding or not. It is theorized that radiation driven meridional flows would lead to an inward spread of this region over the life time of the sun. However, it is also theorized that the turbulence created by even weak magnetic field creates a viscous force that acts like friction to prevent this spreading.

CHAPTER 3

Methods

Our goal is to regenerate the results of this paper by running computational simulations on Bates College's high-performing computing cluster(HPCC). The HPCC is named after the twentieth century astronomer Henrietta Swan Leavitt. She is most notably known for discovering the relation between luminosity and the period of Cepheid variable stars. Leavitt includes sixteen Intel Xeon compute nodes with a total of 448 cores, plus a GPU node with dual Intel Xeon Phi processors, each with 64 cores. Each core can run one programming command instruction, and the multicore setup allows for several tasks to be run simultaneously. We are able to communicate with Leavitt remotely from my windows desktop which is running a virtual Ubuntu machine. Leavitt itself is running Linux CentOS 7 operating system.

The program used to write our scripts is the text editor Emacs. These simulations have two files included with them, one Python script thats solves the system of differential equations and one run script which instructs Leavitt on how to run the Python script. Data is output to .h5 files which are merged together at the end of the simulation. We then use a third script to plot the data.

Dedalus is a spectral method differential equation package for python built by my advisor Jeffrey Oishi and his colleagues. It uses bases in order to generate sparse matrices that lead to solutions. Dedalus has a few bases that the user can pick from when writing code. However, for the purpose of this research we will use the Fourier basis. One disadvantage the Fourier series has lies within the nature of its bounds. This basis requires the beta plane to be periodic. However, graphing the linear beta plane periodically elicits a discontinuous function shown on the next page.

Therefore, we must adapt our equations for the y component in order to create a stable, periodic, continuous form of the Coriolis force. To do this we use sponge layers which will give

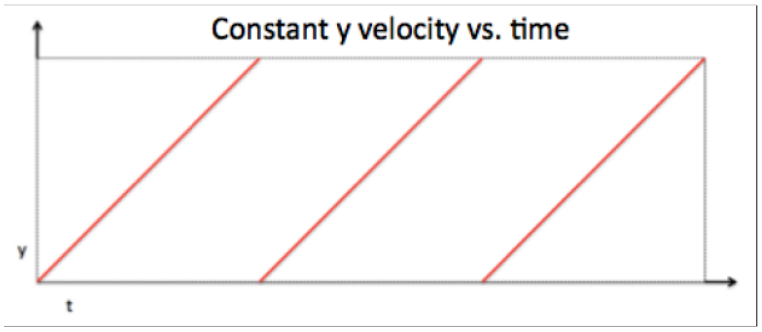


FIGURE 3.1. Graph of beta plane showing discontinuities at peaks and troughs of y

us inaccurate results at the edges of the simulation however this does not effect the accuracy of the simulation near the equator. To do this we introduced new equations for y where Y(y) is the control equation and Z(y) is the damping equation.

$$Y(y) = \frac{L_y(1+\zeta)}{\pi\zeta} arctan\left(\frac{\zeta sin(\theta)}{1+\zeta cos(\theta)}\right) \tag{3.1}$$

$$Z(y) = \frac{(1-\zeta)^2}{2}\frac{(1-cos(\theta))}{(1+\zeta^2+2\zeta cos(\theta))} \tag{3.2}$$

In these equations ζ is the control parameter for the sponge layer, L_y is the length in longitude, and $\theta = \frac{\pi}{L_y}$ These adjustments allow us to run the scripts using a Fourier series basis with the equatorial beta plane being both periodic and continuous.

The equations for this model are the forced momentum equation for incompressible fluid combined with the magnetic induction equation.

$$\frac{\delta u}{\delta t} + (u \cdot \nabla) + 2\Omega \times u = -\nabla p + j \times B + v\nabla^2 u + F \tag{3.3}$$

$$\nabla \cdot u = 0 \tag{3.4}$$

$$\frac{\delta B}{\delta t} = \nabla \times (u \times B) + \eta \nabla^2 B \tag{3.5}$$

Where u is the velocity, F is the forcing, Ω is the rotation, B is the magnetic field, j is the current and v and η are non-dimensional measures of the viscosity and magnetic diffusivity. Now we consider a local domain in the lower tachocline using x and y coordinate because our two dimensional model leaves all variables independent of z. substitutions are then made for B and yield the following equations solved using Dedalus on the supercomputer.

$$\omega_t = J(\Psi, \omega) + \beta \Psi_x + J(A, \nabla^2 A) - B_0 \nabla^2 A_x + v \nabla^2 \omega + G_0 \tag{3.6}$$

$$A_t = J(\Psi, A) + B_0 \Psi_x + \eta \nabla^2 A \tag{3.7}$$

Where A is the vector potential and ω is the vorticity. The forcing G_0 is designed to sample spectral modes with wavenumbers $14 \leq k_x, k_y \leq 20$. We do not include any sink term to remove energy at large scales. This is because large scale zonal flows would dominate the small scale turbulence and we do not wish to suppress the magnetic field.

Using these equations the researchers involved in β-Plane Magnetohydrodynamic Turbulence in the Solar Tachocline were able to show how even small magnetic fields ranging between $10^-4 \leq B_0 \leq 10^-1$ disturbed the zonal flows and created turbulence within the simulation. The researchers first created the plot below showing the zonal flows with no magnetic field applied.

Next they ran the simulations with weak magnetic fields applied and created the following plots which show the zonal flows devolving into turbulence.

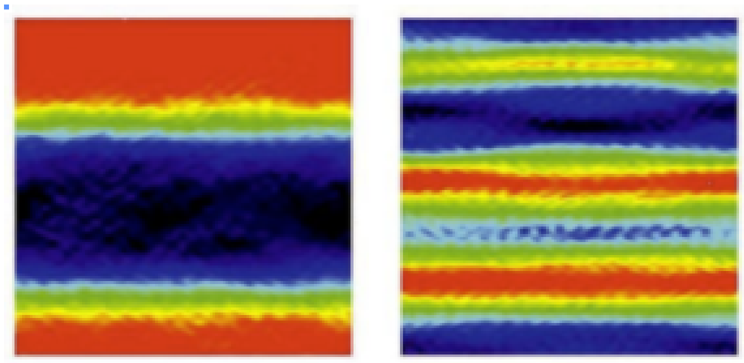


FIGURE 3.2. Color-coded density plots Red indicates strong pro-grade flows, while blue indicates strong flow field

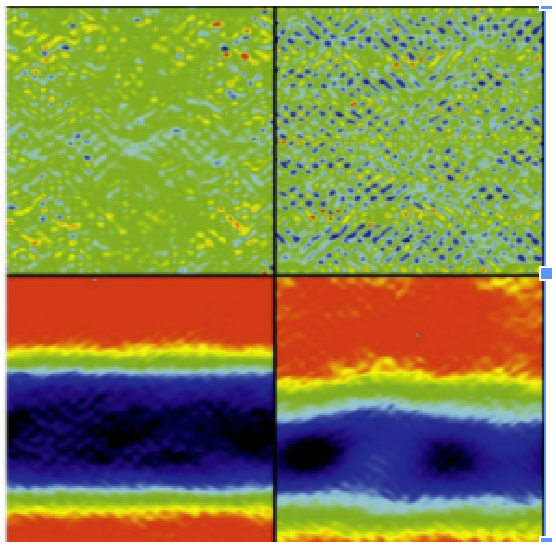


FIGURE 3.3. Color-coded density plots that show magnetic field creating turbulence

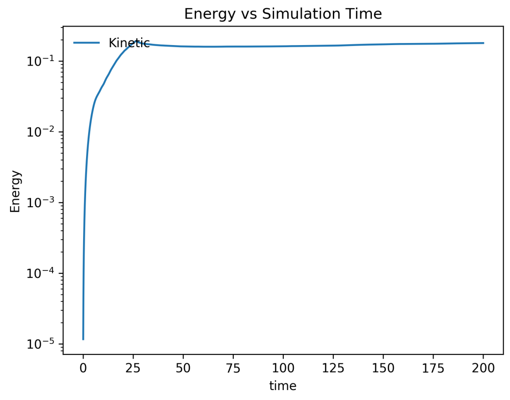
CHAPTER 4

Results

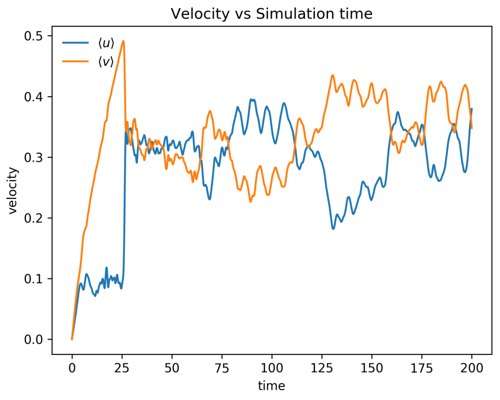
In this section we will explore the simulations of two cases. Both of these cases use the equatorial beta plane in simulating Rossby waves in the tachocline region of the sun. The first case was performed to check the viability of our simulations. We ran our code with no magnetic field present and the mhd parameter turned off. The second case we instead apply magnetic fields of varying strength and activate the mhd parameter within the code.

1. Without Magneto-hydrodynamic Turbulence

The first simulations that we created were meant to see if our script ran properly and output the correct data. The parameters for this script were; $\beta = 0$, $B_0 = 0$, and mhd turned off. This code ran for 123057 iterations over a simulation time of 200 which is a unit-less parameter. This translated to roughly 12.86 hours in real time. Using the data output by our script we then plotted kinetic energy over the course of the simulation and we also plotted the root mean square of the u and v velocities.



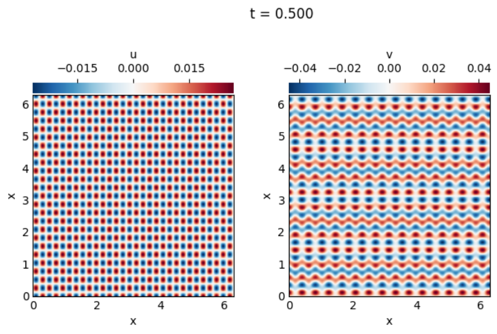
(A) Kinetic Energy vs Simulation Time



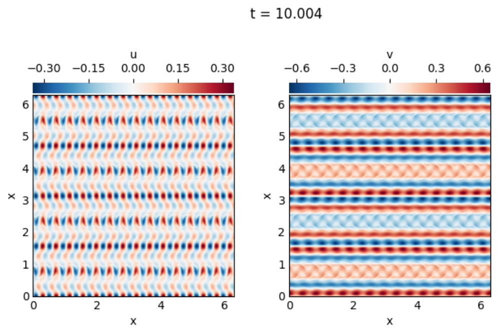
(B) Velocity vs Simulation Time

FIGURE 4.1. Plots from simulation without magnetic field applied

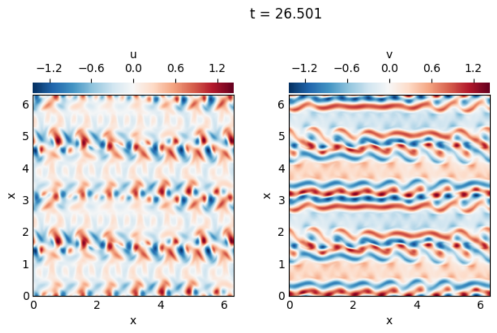
We can see in these plots the initial kick of energy that we give the system, that is why both plots start with y values of zero. Units for energy are dimensionless due to the original setup of the partial differential equations and velocity is given in meter per second. Next we run another plotting script that plots u and v velocities using a standard red blue color map to show the development of zonal flows over time and how they devolve into turbulence.



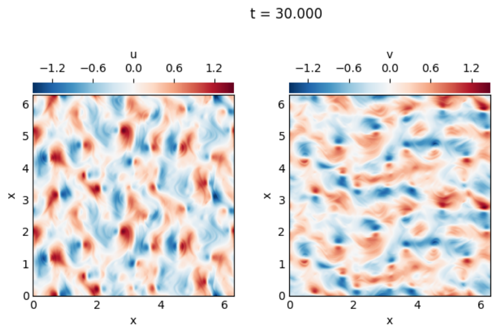
(A) Velocities at T=0.500



(B) Velocities at T=10.004



(C) Velocities at T=26.501



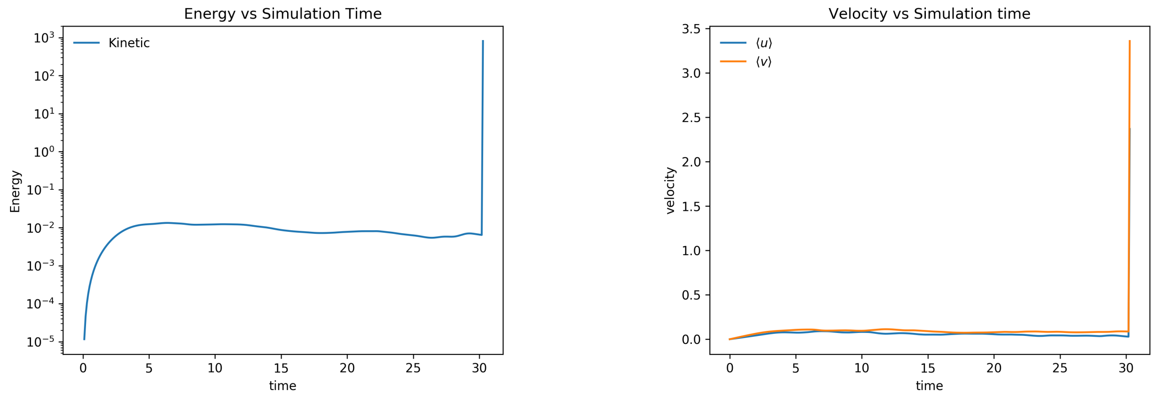
(D) Velocities at T= 30.000

FIGURE 4.2. Four frames at varying times with no magnetic field

The first of these frames shows the system at time, T=0.5. In this frame we see a stable system first starting to generate zonal flows. The next frame at T=10.001 shows strong zonal flows which then begin to devolve into turbulence in the following frame at T=26.501. Finally we see in the last frame a turbulent system with no zonal flows. These frames show that our script is capable of producing data that we are able to plot and shows that in general our simulation is working. The next step we took was to run the code with our mhd parameter turned on and applied varying magnetic fields.

2. Varying Magnetic Fields

The next step we took was to run the simulation again with the mhd parameter turned on and at varying strengths of magnetic fields. We chose to run the code with $B_0 = 1 \times 10^{-3}, 1 \times 10^{-2}, 1 \times 10^{-1}$. Our initial results showed we needed to run the code with a longer stop time in order to give enough time for the zonal flows to devolve into turbulence. Both simulations with $B_0 = 1 \times 10^{-1} and 1 \times 10^{-3}$ ran until simulation time of 100.0 without devolving into turbulence, however the simulation with $B_0 = 1 \times 10^{-2}$ crashed at around simulation time 30.0. The plots for the crashed simulation below indicated to us that the crash was due to a time stepping error so we added a new function that measured velocities and adjusted the time step taken in between iterations accordingly.



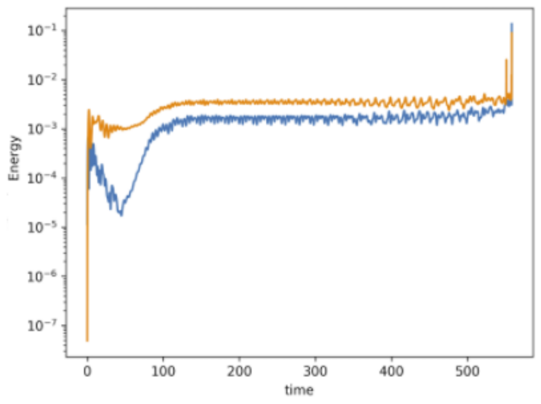
(A) Kinetic Energy vs Simulation Time (B) Velocity vs Simulation Time

FIGURE 4.3. Plots that show crash with $B_0 = 1 \times 10^{-2}$

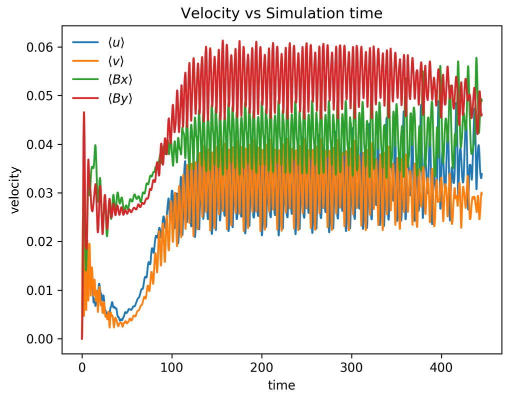
As we can see in these plots the energy and velocities sky rocketed to infinity at roughly simulation time 30. A smaller time step can be used to fix this issue however that is more computationally taxing. In order to achieve this we added a CFL function which added the u and v velocities after every iteration and determined the best time step to make sure the values did not change too quickly.

Our next step after including the CFL function was to run the simulations that did not crash with larger stop times in order to see the system devolve into turbulence. We chose to use a stop time of 1000.0. Now the simulations ran for anytime between 24 to 48 hours in order to reach completion. This resulted in two new complications faced. First our simulation with

$B_0 = 1 \times 10^-1$ begin crashing at simulation time roughly 550 despite the new CFL function having been added. This prompted us to plot magnetic energy as well as kinetic energy. Also as a result of the initial equations magnetic field in the x and y directions were given in units of velocity which allowed us to plot magnetic fields and compare them to the velocities of the waves in the u and v directions.



(A) Kinetic Energy vs Simulation Time



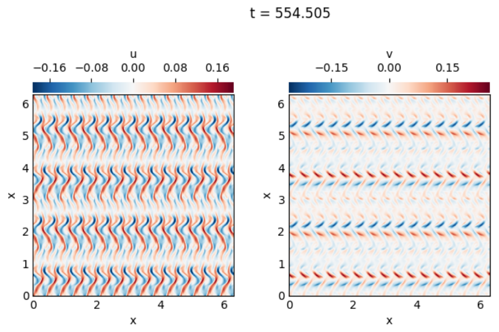
(B) Velocity vs Simulation Time

FIGURE 4.4. Plots that show crash with $B_0 = 1 \times 10^{-1}$

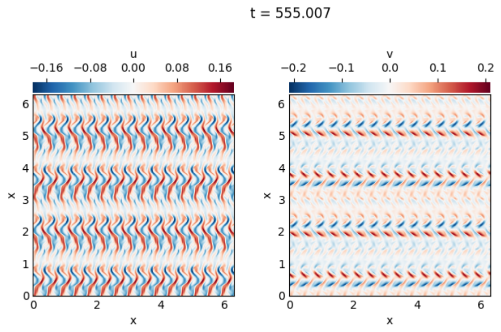
As we can see in these plots magnetic velocities in the x and y directions are greater than the u and v velocities. This explains why magnetic energy is greater than kinetic energy in the plot on the left. This led us to use B_x and B_y values instead of u and v within the CFL function. We then looked at the frames right around where the crash was happening to see if those gave any further evidence as to what could have been causing the crash.

The frames below show the crash happens very quickly with the upper left plot still showing stable zonal flows at T = 554.505 and then a complete crash just a few frames later at T = 556.500 shown in the bottom right plots. This is further evidence of a time stepping error.

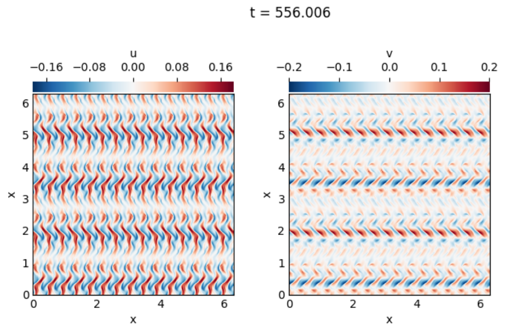
Furthermore we were given more proof that the error must be a time stepping problem by observing the info log that can be output to the Linux command line while the simulation is running. Every ten iterations a statement was printed to the command line that included variables such as dt, urms, and vrms. At approximately 2000 iterations before the crash our dt parameter begins changing dramatically by multiple factors of ten between each set of ten



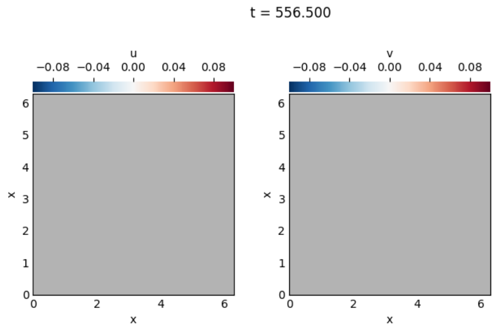
(A) Velocities at T=554.505



(B) Velocities at T=555.007



(C) Velocities at T=556.006



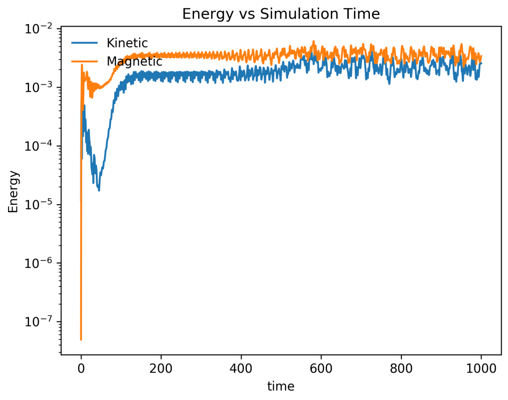
(D) Velocities at T= 556.500

FIGURE 4.5. Four frames at varying times showing crash with $B_0 = 1 \times 10^{-1}$

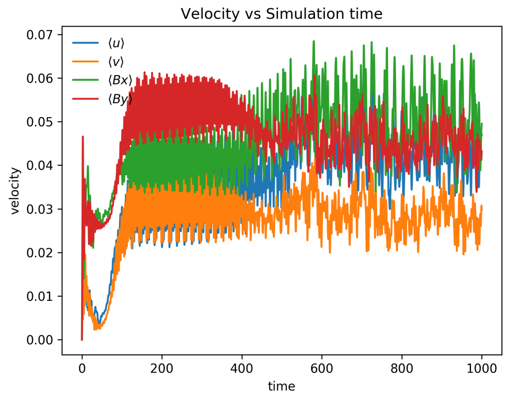
iterations. This was due to the CFL function attempting to find a small enough time step to stabilize the system and keep it from crashing.

The next step we took to resolve this problem was to add another line to the CFL function that forced the function to consider all four velocities; u, v, B_x, B_y when calculating time step needed. This allowed the CFl function to work properly and calculate the appropriate time step to keep the simulations from crashing. Below is the plotted data with $B_0 = 1 \times 10^{-1}$ run with a simulation stop time of 1000.

We can see in these plots that the simulation reaches simulation time of 1000 without energy spiking to infinity. The point where the CFL function kicks in to maintain an appropriate time step is also visible in these plots. We have further evidence that the time stepping error was officially fixed when we examine the frames below which show the simulation running to completion.

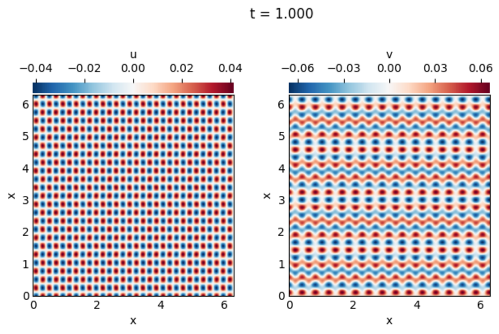


(A) Kinetic Energy vs Simulation Time

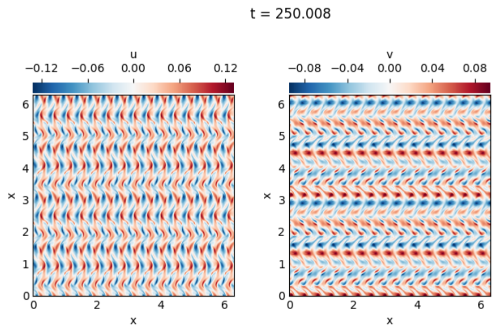


(B) Velocity vs Simulation Time

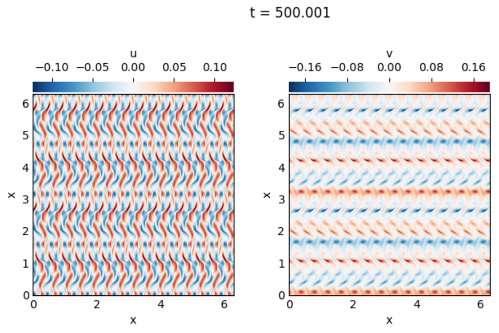
FIGURE 4.6. Plots that show simulation completing with $B_0 = 1 \times 10^{-1}$



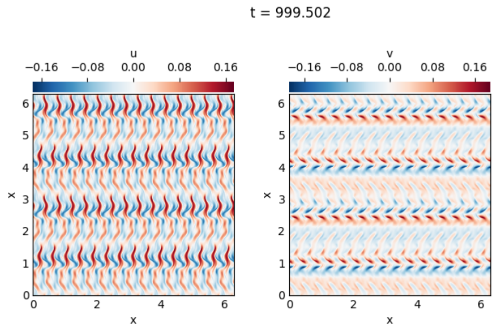
(A) Velocities at T=1.000



(B) Velocities at T=250.008



(C) Velocities at T=500.001



(D) Velocities at T= 999.502

FIGURE 4.7. Four frames at varying times showing CFL function works $B_0 = 1 \times 10^{-1}$

CHAPTER 5

Discussion

This thesis has demonstrated the effective use of the periodic equatorial beta plane and how it can be used to mathematically describe fluid dynamics in the Tachocline region of the sun. We have managed to show turbulence in our simulations without magnetic fields. Also we have demonstrated the effects varying strengths of magnetic fields have on Rossby waves propagating in the lower Tachocline region.

We managed to use Fourier series to create a periodic Beta Plane substituting y with sponge layers to make the Fourier series stable. From there we created computational simulations to solve a set of partial differential equations and used the output data to plot kinetic energy and velocity vs time. In order to achieve this we used python framework Dedalus and the HPCC Leavitt here on Bates campus to run our simulations and map the behavior of these Rossby waves.

The results of this test case is viable evidence that both the equatorial beta plane and Dedalus can be used to model Rossby waves in the Tachocline region of the Sun and how they are effected by even low strength magnetic fields. Future research is needed to explore what happens when we drop the y value to be closer to the equator of the sun to analyze if the equatorial beta plane still works properly for waves at or near the equator. Another interesting expansion would be to look into the possibilty of creating three dimensional models so as to better understand this region and see the viability of modeling fluid dynamics in three dimensions.

Appendix A

Script for Alfven Nonequatorial Beta Plane

```
"""alfven_nonequatorial_beta_plane_A_omega

Run the driven non-equatorial beta plane equations from Tobias, Diamond, & Hughes
    (2007).

Usage:
    alfven_nonequatorial_beta_plane_A_omega.py [--Lx=<Lx> --Ly=<Ly> --nx=<nx> --
        ny=<ny> --mhd --B0=<B0> --beta=<beta> --eta=<eta> --nu=<nu> --dt=<dt> --
        G0=<G0> --n_modes=<n_modes>]

Options:
    --Lx=<Lx>                                  length in latitude [default:
        6.283185307179586]
    --Ly=<Ly>                                  length in longitude [default:
        6.283185307179586]
    --nx=<nx>                                  grid points in x [default: 128]
    --ny=<ny>                                  grid points in y [default: 128]
    --B0=<B0>                                  Zonal (x) background field strength
        [default: 0]
    --mhd                                      Use MHD
    --beta=<beta>                              beta parameter [default: 5]
    --nu=<nu>                                  viscosity [default: 1e-4]
    --eta=<eta>                                diffusivity [default: 1e-4]
    --G0=<G0>                                  forcing amplitude [default: 2.0]
    --n_modes=<n_modes>                          number of modes to force with [
        default: 3]
"""
import sys
import os
import numpy as np
from mpi4py import MPI
import time
from dedalus.tools.config import config
from dedalus.extras import flow_tools

from docopt import docopt

# parse arguments for parameters
args = docopt(__doc__)

Lx = float(args['--Lx'])
Ly = float(args['--Ly'])
nx = int(args['--nx'])
ny = int(args['--ny'])
beta = float(args['--beta'])
nu = float(args['--nu'])
B0 = float(args['--B0'])
eta = float(args['--eta'])
G0 = float(args['--G0'])
n_modes = int(args['--n_modes'])

if args['--mhd']:
    mhd=True
else:
    mhd=False
```

```

basename = sys.argv[0].split('.py')[0]

data_dir = "scratch/" + basename
data_dir += "_Lx{0:f}_Ly{1:f}_nx{2:d}_ny_{3:d}_beta{4:5.02e}_B0{5:5.02e}_nut
    {6:5.02e}".format(Lx,Ly,nx,ny,beta,B0,nu)

# if cheb:
#     data_dir += "_chebyshev"

config['logging']['filename'] = os.path.join(data_dir,'dedalus_log')
config['logging']['file_level'] = 'DEBUG'

import dedalus.public as de
from dedalus.extras import flow_tools
from dedalus.tools import post
import logging
logger = logging.getLogger(__name__)

# Create bases and domain
# bases
x_basis = de.Fourier('x', nx, interval=(0,Lx), dealias=3/2)
y_basis = de.Fourier('y', ny, interval=(0, Ly), dealias=3/2)

domain = de.Domain([x_basis,y_basis], grid_dtype=np.float64)

# 2D Boussinesq magnetohydrodynamics on the beta plane
variables = ['omega','psi']
if mhd:
    variables.append('A')
bp = de.IVP(domain,variables=variables)
```

```
bp.parameters['beta'] = beta
bp.parameters['B0'] = B0
bp.parameters['Lx'] = Lx
bp.parameters['Ly'] = Ly
bp.parameters['pi'] = np.pi
bp.parameters['eta'] = eta
bp.parameters['nu'] = nu
bp.parameters['G0'] = G0
bp.parameters['n_modes'] = n_modes

bp.substitutions['vol_avg(A)']    = 'integ(A)/(Lx*Ly)'
bp.substitutions['J(A,B)'] = 'dx(A)*dy(B) - dx(B)*dx(A)'
bp.substitutions['Lap(A)'] = 'dx(dx(A)) + dy(dy(A))'
bp.substitutions['u'] = 'dy(psi)'
bp.substitutions['v'] = '-dx(psi)'

def forcing(*args):
    x = args[0].data
    y = args[1].data
    ampl = args[2].value
    n_modes = args[3].value

    kx = [14, 12, 12]
    ky = [0, 15, -15]
    phase_x = [0, 0, 0]
    phase_y = [0, 0, 0]

    spatial_pattern = 0
    for i in range(n_modes):
        spatial_pattern += np.cos(kx[i]*x + phase_x[i])*np.cos(ky[i]*y + phase_y[
            i])/n_modes

    return ampl*spatial_pattern

def GF(*args, domain=domain, F=forcing):
    return de.operators.GeneralFunction(domain, layout='g', func=F, args=args)

de.operators.parseables['G'] = GF

if mhd:
    bp.add_equation("dt(omega) + B0*Lap(dx(A)) - nu*Lap(omega) = J(psi,omega) +
        beta*dx(psi) + J(A,Lap(A)) + G(x, y, G0, n_modes)")
    bp.add_equation("dt(A) - B0*dx(psi) - eta*Lap(A) = J(psi, A)")
else:
    bp.add_equation("dt(omega) - nu*Lap(omega) = J(psi,omega) + beta*dx(psi) + G(
        x, y, G0, n_modes)")
bp.add_equation("omega + Lap(psi) = 0", condition='nx != 0 or ny != 0')
bp.add_equation("psi = 0", condition='nx == 0 and ny == 0')

# Build solver
dt = 0.01
ts = de.timesteppers.RK443
IVP = bp.build_solver(ts)
IVP.stop_sim_time = 100.
IVP.stop_wall_time = np.inf
IVP.stop_iteration = 10000000
logger.info('Solver built')


CFL = flow_tools.CFL(IVP, initial_dt=dt, cadence=10, safety=0.8, max_change=2.0)
CFL.add_velocities(('u','v'))

```

```
if domain.distributor.rank == 0:
    if not os.path.exists('{:s}/'.format(data_dir)):
        os.mkdir('{:s}/'.format(data_dir))

# Analysis
snapshots = IVP.evaluator.add_file_handler(os.path.join(data_dir,'snapshots'), sim_dt=1., max_writes=50)
snapshots.add_system(IVP.state)

slices = IVP.evaluator.add_file_handler(os.path.join(data_dir,'slices'), sim_dt=0.5, max_writes=50)
slices.add_task("u",name="u")
slices.add_task("v",name="v")

timeseries = IVP.evaluator.add_file_handler(os.path.join(data_dir,'timeseries'), iter=10, max_writes=np.inf)
timeseries.add_task("vol_avg(sqrt(u*u))",name='urms')
timeseries.add_task("vol_avg(sqrt(v*v))",name='vrms')
timeseries.add_task("vol_avg(0.5*(u*u+v*v))",name='Ekin')
timeseries.add_task("vol_avg((u*u)/(u*u+v*v))",name='Ekin_x_ratio')

analysis_tasks = [snapshots, slices, timeseries]

flow = flow_tools.GlobalFlowProperty(IVP, cadence=10)
flow.add_property("0.5*(u*u + v*v)", name="Ekin")
flow.add_property("u", name="u")
flow.add_property("v", name="v")
# Main loop
try:
    logger.info('Starting loop')
    start_time = time.time()
    while IVP.ok:
        dt = CFL.compute_dt()
        dt = IVP.step(dt)
        if (IVP.iteration-1) % 10 == 0:
            logger.info('Iteration: %i, Time: %e, dt: %e, Kinetic Energy: %e, u max: %e, v max: %e' %(IVP.iteration, IVP.sim_time, dt, flow.volume_average('Ekin'), flow.max('u'), flow.max('v')))
except:
    logger.error('Exception raised, triggering end of main loop.')
    raise
finally:
    end_time = time.time()
    logger.info('Iterations: %i' %IVP.iteration)
    logger.info('Sim end time: %f' %IVP .sim_time)
    logger.info('Run time: %.2f sec' %(end_time-start_time))
    logger.info('Run time: %f cpu-hr' %((end_time-start_time)/60/60*domain.dist.comm_cart.size))

    for task in analysis_tasks:
        logger.info(task.base_path)
        post.merge_analysis(task.base_path)
```

Appendix A1

Run script with no mhd parameter or magnetic field

```
#!/usr/bin/bash
#SBATCH --nodes=2

source /home/joishi/build/dedalus_intel_mpi/bin/activate

date
mpirun -np 32 python3 alfven_nonequatorial_beta_plane_A_omega.py --nx=1024 --ny
    =1024 --beta=0.
date
```

Appendix A2

Plot script for Energy vs Time and Velocity vs Time

```
import matplotlib
matplotlib.use('Agg')
import sys
import h5py
import matplotlib.pyplot as plt


fname = sys.argv[1]
data = h5py.File(fname,'r')

plt.plot(data['/scales/sim_time'],data['/tasks/urms'][:,0,0],label=r'$\left< u \
    right>$')
plt.plot(data['/scales/sim_time'],data['/tasks/vrms'][:,0,0],label=r'$\left< v \
    right>$')
plt.plot(data['/scales/sim_time'],data['/tasks/Bx'][:,0,0],label=r'$\left< Bx \
    right>$')
plt.plot(data['/scales/sim_time'],data['/tasks/By'][:,0,0],label=r'$\left< By \
    right>$')
plt.title('Velocity vs Simulation time')
plt.legend(loc='upper left').draw_frame(False)
plt.xlabel("time")
plt.ylabel("velocity")

plt.savefig('rms_vel_vs_t.png',dpi=300)
plt.clf()
```

```
#fig = plt.figure()
#ax1 = fig.add_subplot(121)
plt.semilogy(data['/scales/sim_time'],data['/tasks/Ekin'][:,0,0],label='Kinetic')
plt.semilogy(data['/scales/sim_time'],data['/tasks/Mag_en'][:,0,0],label='
    Magnetic')
plt.title('Energy vs Simulation Time')
#plt.legend(loc='upper left').draw_frame(False)
plt.xlabel("time")
plt.ylabel("Energy")
plt.legend(loc='upper left').draw_frame(False)

###ax2 = fig.add_subplot(122)
###ax2.plot(data['/scales/sim_time'],data['/tasks/Ekin_x_ratio'][:,0,0])
###ax1.set_xlabel("time")
###ax2.set_ylabel(r"$u^2/(u^2 + v^2)$")
plt.savefig('energy_xfrac_vs_t.png',dpi=300)
```

Appendix A3

Plot script to create frames of the data

```
"""
Plot planes from joint analysis files.

Usage:
    plot_2d_series.py <files>... [--output=<dir>]

Options:
    --output=<dir>  Output directory [default: ./frames]

"""

import h5py
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
plt.ioff()
from dedalus.extras import plot_tools


def main(filename, start, count, output):
    """Save plot of specified tasks for given range of analysis writes."""

    # Plot settings
    #tasks = ['psi', 'omega']
    tasks = ['u', 'v']
    scale = 2.5
    dpi = 100
    title_func = lambda sim_time: 't = {:.3f}'.format(sim_time)
    savename_func = lambda write: 'write_{:06}.png'.format(write)
    # Layout
    nrows, ncols = 1, 2
    image = plot_tools.Box(1, 1)
    pad = plot_tools.Frame(0.2, 0.2, 0.1, 0.1)
    margin = plot_tools.Frame(0.3, 0.2, 0.1, 0.1)

    # Create multifigure
    mfig = plot_tools.MultiFigure(nrows, ncols, image, pad, margin, scale)
    fig = mfig.figure
    # Plot writes
    with h5py.File(filename, mode='r') as file:
        for index in range(start, start+count):
            for n, task in enumerate(tasks):
                # Build subfigure axes
                i, j = divmod(n, ncols)
                axes = mfig.add_axes(i, j, [0, 0, 1, 1])
                # Call 3D plotting helper, slicing in time
                dset = file['tasks'][task]
                image_axes = (1, 1)
                data_slices = [index, slice(None), slice(None)]
                plot_tools.plot_bot(dset, image_axes, data_slices, axes=axes,
                    title=task, even_scale=True)
            # Add time title
            title = title_func(file['scales/sim_time'][index])
            title_height = 1 - 0.5 * mfig.margin.top / mfig.fig.y
            fig.suptitle(title, x=0.48, y=title_height, ha='left')
            # Save figure
            savename = savename_func(file['scales/write_number'][index])
            savepath = output.joinpath(savename)
            fig.savefig(str(savepath), dpi=dpi)
            fig.clear()
    plt.close(fig)


if __name__ == "__main__":

    import pathlib
    from docopt import docopt
    from dedalus.tools import logging
    from dedalus.tools import post
    from dedalus.tools.parallel import Sync

    args = docopt(__doc__)

    output_path = pathlib.Path(args['--output']).absolute()
    # Create output directory if needed
    with Sync() as sync:
        if sync.comm.rank == 0:
            if not output_path.exists():
                output_path.mkdir()
    post.visit_writes(args['<files>'], main, output=output_path)
```

Appendix A4

Run script with mhd turned on and $B_0 = 1 \times 10^{-1}$

```
#!/usr/bin/bash
#SBATCH --nodes=2 --time=4200

source /home/joishi/build/dedalus_intel_mpi/bin/activate

date
mpirun -np 32 python3 alfven_nonequatorial_beta_plane_A_omega.py --nx=1024 --ny
    =1024 --beta=5. --mhd --B0=1e-1
date
```

Appendix A5

Merge script that combines output .h5 files

```
"""
Merge distributed analysis sets from a FileHandler.

Usage:
    merge.py <base_path> [--cleanup]

Options:
    --cleanup   Delete distributed files after merging

"""

if __name__ == "__main__":

    from docopt import docopt
    from dedalus.tools import logging
    from dedalus.tools import post

    args = docopt(__doc__)
    post.merge_analysis(args['<base_path>'], cleanup=args['--cleanup'])
```

Bibliography