Reconnection-Intro

September 30, 2022

1 Motivation

In this module we will discuss current computational methods to modeling plasmas. In particular, we will simulate an ubiquitous phenomena in laboratory, space, and astrophysical plasmas called magnetic reconnection. Additionally, in this module we will introduce a broad array of skills useful in computational physics which you may not have seen before: Git version control, shell commands, post-processing simulation data using Python. A variety of plasma models will be employed which span the space of both fidelity and computational complexity, from first-principles kinetic modeling to approximate fluid models like magnetohydrodynamics. We will aim to properly introduce these vital skills to students, taking care to cater to various levels of prior knowledge. (**figures demonstrating high performance computing**)

The file you are viewing is called a Jupyter Notebook, it is a cell based multiplatform file that can host live code from different languages, TeX equations, multimedia and text all in one file. All of the former options can be run in a cell with SHIFT+ENTER. If you have joined here from the Github README, go to the setup procedure for your respective operating system - Windows or macOS. If you are using the former, the procedure will take you through creating a linux virtual machine and installing this notebook file and the rest of the folder in that virtual machine. If you are using Macos, you likely have this repository downloaded, so the setup procedure will be shorter.

Jupyter Lab automatically creates a Table of contents that you can access by clicking this icon on the sidebar to navigate the notebook. Remember to look at the README file as well for a overview of the repository and files contained.

2 Background

2.1 An overview of magnetic reconnection

Magnetic reconnection is a process in which the magnetic topology is rearranged in the presence of a plasma. This topological rearrangement of the field lines is a mechanism of converting magnetic field energy to other forms of energy: heat, large scale plasma flows, and accelerating particles to very high energies. It is easiest to visualize this process with a pair of oppositely directed (antiparallel) magnetic field lines in a system that experiences high heat levels. One can think of the magnetic field lines snapping and "reconnecting" with other field lines in complete analogy with a rubber band snapping (*see the gif below*). Common examples of this are coronal mass ejections and when the solar wind impacts the Earth's magnetosphere. Magnetic field lines have inherent

tension, which act to straighten out magnetic field lines. When reconnection occurs, this force acts towards the center of curvature and pushes the plasma out, similar to a slingshot. We will discuss the source of tension later, but first we will find out what, mathematically, causes this process.

Source: By ChamouJacoN - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=7496329

2.2 A deeper dive into magnetic reconnection

We first start with the ideal Ohm's Law equation for ideal non-resistive magnetohydrodynamic models

$$\mathbf{E} + \mathbf{V} \times \mathbf{B} = 0$$

which describe systems that follow the frozen in theorem. The theorem states that without any resistivity in a fluid, the magnetic field is frozen in to said fluid. In a plasma this means that the magnetic field lines move with the plasma. Furthermore, the field will not change when moving along with the field lines, but motions transverse to the field carry the field with the motion.

We can derive the frozen-in theorem by taking Ohm's Law and applying it to Faraday's Law, getting

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{V} \times \mathbf{B}) - c\nabla \times (\eta \mathbf{J})$$

since the current density J is proportional to $\nabla \times \mathbf{B}$ through Ampere's Law, we arrive at

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{V} \times \mathbf{B}) + c\eta(\nabla^2 \mathbf{J})$$

after some simplifications, namely using the curl of a curl identity and that $\nabla \cdot B = 0$ since there are no magnetic monopoles.

Since $\eta = \frac{1}{\mu_0 \sigma}$, where μ_0 is the permiability of free space as σ is the electrical conductivity, when we look no resistivity/infinite conductivity, the η term in the magnetic field equation falls out and we are left with the induction equation.

With the induction equation

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{V} \times \mathbf{B})$$

we use $\Phi = \int_s \mathbf{B} \cdot dS$ to get

$$\frac{\mathrm{d}\Phi}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t} \int_s \mathbf{B} \cdot dS$$

The two ways the magnetic flux can change is through a change in the magnetic field strength $\frac{\partial B}{\partial t}$ or a movement in the surface we integrate over. We can substitute the induction equation in for the first term and get

$$\int_s \nabla \times (\mathbf{V} \times \mathbf{B}) \cdot ds + \int_s \mathbf{B} \cdot (\mathbf{V} \times d\ell)$$

The second term arises from movement on the surface of integration. We can use Stokes' Theorem and the vector identity $(A \times B) \cdot C = -B \cdot (A \times C)$ to get

$$\int_c \mathbf{V} \times \mathbf{B} \cdot d\ell - \int_c (\mathbf{V} \times \mathbf{B}) \cdot d\ell = 0$$

Thus, we find that $\frac{d}{dt}\int_s \mathbf{B}\cdot ds = 0$, proving the frozen in theorem when we have no resistivity.

When we introduce resistivity, Ohm's Law transforms into

$$\mathbf{E} + \mathbf{V}\times\mathbf{B} = \eta\mathbf{J}$$

where η is the magnetic diffusivity, and the system no longer follows the frozen in theorem. This means that the magnetic field lines are no longer frozen to the plasma and can reconnect.

2.3 Modeling magnetic reconnection

In order to talk about the models we must find all the physical properties to consider when generating a model and any approximations. First we start with the electron momentum equation

$$m\frac{\partial n\mathbf{u_e}}{\partial t} + \nabla\cdot\overleftrightarrow{\mathcal{P}} = nq(\mathbf{E} + \mathbf{u_e}\times\mathbf{B})$$

and our goal is to end up with the generalized Ohm's Law equation which will play a key role in the future. In order to get there, we rearrange the electron momentum equation to get $E = \frac{m}{nq}\frac{\partial n\mathbf{u_e}}{\partial t} + \nabla\cdot\overleftrightarrow{\mathcal{P}} - \mathbf{u_e}\times\mathbf{B}$. The u in the term is electron flow, so we can subsitute in $\mathbf{u_e} = -\frac{\mathbf{J}}{q_s n_s} + \mathbf{u_i}$ where J is current density and $\mathbf{u_i}$ is ion flow. With this information we get

$$-\mathbf{u_e}\times\mathbf{B} = (\frac{\mathbf{J}}{q_s n_s} - \mathbf{u_i})\times\mathbf{B} = \frac{\mathbf{J}\times\mathbf{B}}{q_s n_s} - \mathbf{u_i}\times\mathbf{B}$$

For the $\overleftrightarrow{\mathcal{P}}$ (stress tensor) term, we can substitute in the equation of the tensor involving reynolds stress

$$\overleftrightarrow{\mathcal{P}} = \overleftrightarrow{\mathbf{P}}_e + \rho\mathbf{u_e}\mathbf{u_e}$$

Now we are left with

$$E + \mathbf{u_i}\times\mathbf{B} = \frac{\mathbf{J}\times\mathbf{B}}{nq} + \frac{1}{nq}\frac{\partial nm\mathbf{u_e}}{\partial t} + \frac{\nabla\cdot\overleftrightarrow{\mathbf{P}}_e}{nq} + \frac{\nabla\cdot nm\mathbf{u_e}\mathbf{u_e}}{nq}$$

We can combine the time derivative of momentum and divergence of Reynolds stress using product rule, so

$$\frac{\partial nm\mathbf{u_e}}{\partial t} = nm\frac{\partial\mathbf{u_e}}{\partial t} + \mathbf{u_e}\frac{\partial nm}{\partial t}$$
$$\nabla\cdot(nm\mathbf{u_e}\mathbf{u_e}) = \mathbf{u_e}\nabla\cdot(nm\mathbf{u_e}) + nm\mathbf{u_e}\cdot\nabla\mathbf{u_e}$$

Resulting in

$$nm(\frac{\partial\mathbf{u_e}}{\partial t} + \mathbf{u_e}\cdot\nabla\mathbf{u_e}) + \mathbf{u_e}(\frac{\partial nm}{\partial t} + \nabla\cdot(nm\mathbf{u_e})) = nm(\frac{\partial\mathbf{u_e}}{\partial t} + \mathbf{u_e}\cdot\nabla\mathbf{u_e})$$

since the second term equals 0 due to mass conservation $\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho u) = 0$. Now we can plug in our new expression for the electron inertia contribution, arriving at

$$\mathbf{E} + \mathbf{u_i}\times\mathbf{B} = \eta\mathbf{J} + \frac{\mathbf{J}\times\mathbf{B}}{nq} - \frac{\nabla\cdot\overleftrightarrow{\mathbf{P}}_e}{nq} + \frac{m}{q}[(\frac{\partial\mathbf{u_e}}{\partial t} + \mathbf{u_e}\cdot\nabla\mathbf{u_e})]$$

where the left hand side comes from the ideal MHD, the ηJ, $J\times B$, and P_e terms are known as the resistive dissipation, hall and the pressure tensor terms, respectively. ### 5 moment models In

this computational demo we will start with the 5 moment simulation, which symplifies the pressure tensor down to a scalar. This is Eulers equations shown below, where the force is Lorentz force

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u_s}) = 0$$

$$\frac{\partial \rho \mathbf{u_s}}{\partial t} + \nabla \cdot (\rho \mathbf{u_s} \mathbf{u_s} + p_s \mathbf{I}) = q_s n_s (\mathbf{E} + \mathbf{u_s} \times \mathbf{B})$$

$$\frac{\partial \mathcal{E}_s}{\partial t} + \nabla \cdot ([\mathcal{E}_s + p] \mathbf{u_s}) = q_s n_s \mathbf{u_s} \cdot \mathbf{E}$$

$$\mathcal{E}_s = \frac{1}{2} \rho_s |\mathbf{u_s}|^2 + \frac{p_s}{\gamma - 1}$$

These equations represent the conservation of mass, momentum and energy, respectively, where the total energy is $\mathcal{E}_s$ and the adiabatic index of an ideal fluid γ is 5/3 for hydrogen plasma. Then we will expand the tensor out and compute the 10 moment simulation. Normally, only including the resistive dissipation term causes reconnection to occur far too slowly to only be attributed to the physics. But the hall term and the divergence of the pressure tensor causes the simulations to reconnect much more faster, closer to real life phenomenon. The hall term alone causes the reconnection rate to be comparable to the Alfvén speed, while "at small values of resistivity the dissipation region forms an elongated Sweet-Parker layer and the rate of reconnection is very low, with an inflow velocity v_i into the X line which scales like $v_i = \frac{\delta}{\Delta} v_A << v_A$, where δ and Δ are the width (controlled by resistivity) and length (macroscopic) of the dissipation region, respectively, and v_A is the Alfven velocity" [Birn et al., 2001] as illustrated by the GEM reconnection challenge results.

As mentioned before, 5 moment models simplify the pressure tensor down to a scalar. The 5 moment model is in fact a generalization of the Hall-MHD model (taking Ohm's law until the hall term) which can be found from the 5 moment model by approximating the speed of light to be infinite. By simplifying the pressure tensor, the 5 moment model is significantly easier to run, but can be an inaccurate model outside of the reconnection zone.

3 Setup Procedure - Windows

3.1 Install WSL 2

- follow directions on https://gkeyll.readthedocs.io/en/latest/install.html#note-on-the-windows-linux-subsystem-wsl (Ubuntu installation preferred, the following instructions may not work on other distributions)
- If installing CUDA, use https://docs.nvidia.com/cuda/wsl-user-guide/index.html#cuda-support-for-wsl2
- Install common libraries with `sudo apt install build-essential`
- Run

```
sudo apt update
sudo apt upgrade
sudo apt autoremove
```

- When using Gkeyll and Jupyter Lab in the future, open the ubuntu application from windows, or type "ubuntu" in another terminal application
  - Using windows terminal to do both of these is recommended for tabs, themes and easy configuration

3.2 Install Conda

- (Recommended) If you want to run Jupyter Lab in WSL 2 on your local machine's browser, install a conda distribution if you do not have one
  - Either install Miniconda (lightweight, easier installation) or Anaconda (far more packages, useful if you also need more features for other applications)
    * Miniconda: first download the installer with the command

    ```
    wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
    sudo chmod +x Miniconda3-latest-Linux-x86_64.sh
    ./Miniconda3-latest-Linux-x86_64.sh
    ```

    and follow the prompts. When you have installed Miniconda correctly, jump to Install Jupyter Lab below.
    * Anaconda: follow https://docs.anaconda.com/anaconda/install/linux/ up to step 9 and jump to Install Jupyter Lab below.
- If you want to install the standalone application, follow directions on https://github.com/jupyterlab/jupyterlab-desktop#download
  - Then Run `jupyter-lab --generate-config`
  - Open the generated file and add: `c.NotebookApp.terminado_settings = { 'shell_command': ['C:\\Users\\d3mon\\AppData\\Local\\Microsoft\\WindowsApps\\ubuntu.ex } ` to the end to change the path and restart JL

3.3 Install Jupyter Lab

- First, either open a new terminal or run `source ~/.bashrc`, then activate conda with `conda activate`
- Then update conda with,

```
conda update conda
conda update --all
```

- Create a new conda virtual environment with `conda create --name [NAME] python=3.8` with a name of your choice. If you choose to do this, you must also run `conda activate [NAME]` with the same name as before if you want to use anything you will install in the next steps. If you create the virtual environment correctly, you should see **[NAME]** text before the terminal prompt like this Avoid using a newer python version than 3.8 due to compatability issues.
- Then install Jupyter Lab and dependencies with

```
conda install matplotlib jupyter jupyterlab pip git
conda install -c conda-forge jupyter_contrib_nbextensions
conda update --all
```

- Clone this repository on your machine with `git clone https://github.com/KasaiYuki/gkylnotebooks.git`
- Then launch Jupyter Lab by entering `jupyter lab --no-browser` in the terminal, and copying one of the URLs listed at the end of the output into a browser on your *local* machine, and navigate to the "gkylnotebooks" folder and open this file. It will look something like this.

  Example appearance

This is the recommended layout, with the notebook and terminal open side by side since not everything can be done directly on the notebook. Now that we have this notebook on the virtual machine, we run the rest of the commands from within the notebook. Any cell that starts with "%%bash" or a line that starts with "!" indicates shell commands similar to the ones that let you install packages before. Run these commands as usual, or if you prefer, copy them into a terminal of your choice (without those characters).

3.4 Installing and compiling GkeyllZero

- Clone GkeyllZero the folder you want the repository to be in using the command below (for best performance use something in /home/USER_NAME/..., **NOT** /mnt/...)

[ ]:
```
!git clone https://github.com/ammarhakim/gkylzero.git
```

- Then, install the two main dependencies that G0 needs: cmake and gfortran
  - cmake: follow instructions in https://apt.kitware.com/ and after run the command below‘
  - gfortran: if you don't have gfortran installed with "build-essential", run `sudo apt install gfortran`
- Navigate to the gkylzero directory and follow instructions in the G0 README file at https://github.com/ammarhakim/gkylzero, starting from the "Building on your machine" section

3.5 Installing Postgkyl

- Follow the instructions at https://gkeyll.readthedocs.io/en/latest/install.html#installing-with-conda-preferred-for-non-developers, make sure that you are in the same conda environment you used previously
- When using WSL 2 on Windows 10, by default GUI is not supported. In order to be able to view the pgkyl output plots, install VcXrv (recommended), or Xming if not possible. Follow the instructions at https://gkeyll.readthedocs.io/en/latest/install.html#additional-steps-required-on-windows-10
  - Then, install xterm on WSL by entering `sudo apt install xterm`
  - Whenever you need the GUI in the future, open XLaunch first and disable access control, the XLaunch icon should appear in the icon tray

3.6 Installing Luajit

- Normally, GkeyllZero uses C code to run the simulations, but in this module we will be using lua wrappers for the sake of more efficiently teaching the process. In order to run lua scripts we need to install Luajit.
  - First clone the fully featured Gkeyll version 2 on your machine (WSL if using windows), then run the make dependencies shell file and configure to install only luajit

[ ]:
```
import os
%cd /home/kasaiyuki # Edit path to home directory
!ls
!git clone https://github.com/ammarhakim/gkyl # Clone the repository in the␣
 ↪home directory
```

```
os.chdir('/home/kasaiyuki/gkyl/install-deps') # Edit path to the 'install-deps'␣
 ↪folder
!ls
!./mkdeps.sh --build-adios=no --build-luajit=yes --build-eigen=no # Install␣
 ↪only the Luajit package
```

- Now we need to add the Gkeyll lua files to the PATH. For WSL users, can be done on the bash prompt by editing the .bashrc file (click on "view" on the top tool bar, then click "show hidden items" and find the file in the file browser), and adding `export LUA_PATH="$HOME/gkylsoft/gkylzero/lib/?.lua;;` towards the end of the file. A similar method can be done on different shells as well.
- (Recommended) create an alias/shortcut to use luajit without typing the full path to the executable.
  - In the .bashrc file, add the line `alias run_lua='~/gkylsoft/luajit/bin/luajit'` if you installed luajit to the default directory.
- Then, restart the terminal with `source ~/.bashrc`. Now, whenever you need to run a lua script, just type `run_lua file.lua`, instead of the full path.

4 Setup Procedure - macOS

5 Running a 5 moment model

We will first start by going through a brief demo of running simulations and viewing outputs. In the 5 moment folder of this repository is a premade lua script to run a 5 moment simulation. - Navigate to this folder and use the luajit alias to run the simulation below: `run_lua rt-5m-gem.lua` for example.

[1]:
```
%cd /home/kasaiyuki/gkylnotebooks/5-moment
!~/gkylsoft/luajit/bin/luajit rt-5m-gem.lua
```

```
/home/kasaiyuki/gkylnotebooks/5-moment
B0=     1
Starting GkeyllZero simulation
  tstart: 0.000000e+00. tend: 5.000000e+02
 Step      1 2.0000e-01. Time-step  2.000000e-01
^C
/home/kasaiyuki/gkylsoft/luajit/bin/luajit:
/home/kasaiyuki/gkylsoft/gkylzero/lib/Moments.lua:751: interrupted!
stack traceback:
        /home/kasaiyuki/gkylsoft/gkylzero/lib/Moments.lua:751: in function 'run'
        rt-5m-gem.lua:184: in main chunk
        [C]: at 0x55a047fb4310
```

- After the simulation is done, you will see "Completed in…" in the terminal and new output files will appear in the folder.

Terminal appearance

6 Plotting Initial Conditions

The output of the fluid models will have multiple sets of files based on the lua instructions, each with a different file name ending (and then a number). The file used in this demo outputs the electron fluid moments end in "elc", ion fluid moments end in "ion", and electromagnetic field files end in "field". The number after the suffix corresponds to the frame the file contains. The number of frames is also determined by a variable in the input file called `nFrame`. First we will start by plotting the initial conditions (files ending in "0") using postgkyl. - First make sure you are in the correct conda environment(and if using WSL, that XLaunch is open) - In that folder, type

```
pgkyl rt-5m-gem-field_0.gkyl rt-5m-gem-elc_0.gkyl rt-5m-gem-ion_0.g
kyl select -c 3 plot --fix-aspect -f0 --subplots
```

```
- The `select -c 3` flag outputs only the z axis plots, while `-f0 --subplots` collates all thr
- Explore the pgkyl documentation at https://gkeyll.readthedocs.io/en/latest/postgkyl/main.html
```

[60]:
```
# Used to save to a file and open inside of Jupyter, does not require XLaunch
%cd /home/kasaiyuki/gkylnotebooks/5-moment
!pgkyl rt-5m-gem-field_0.gkyl rt-5m-gem-elc_0.gkyl rt-5m-gem-ion_0.gkyl select␣
 ↪-c 3 plot --fix-aspect -f0 --subplots --saveas '5m-init.png'
```

```
/home/kasaiyuki/gkylnotebooks/5-moment
Figure(640x480)
```

The above command should output an image in the folder, that looks something like this:

With the command above, we plotted ρuz for the electron and ion fluid moments, and the x component of magnetic field, since that drives reconnection. In the field plot, we can see that there is a big difference in the direction of the magnetic field between the top and bottom halves due to the antisymmetric plasma flow. This disparity is also shown in the fluid moment plots, where we can see the midpoint behaving differently. Now that we briefly showed how to view output through the command line, lets take a look at creating python scripts to do the postprocessing. We will be using a tool called pgkylFrontEnd to do so. First, clone the repository below.

[ ]:
```
%cd /home/USERNAME
!git clone https://github.com/jtenbarg/pgkylFrontEnd.git
%cd /home/USERNAME/gkylnotebooks/
```

Then we import the neccessary modules and set up the file paths and variables.

First, copy the data files from the 5-moment folder to pgkylFrontEnd and edit the file paths below for your situation.

[11]:
```
import numpy as np
# Moves the folder to look for modules in
%cd /home/kasaiyuki/pgkylFrontEnd/
from utils import gkData
import matplotlib.pyplot as plt
# import scipy as sp
params = {} # Initialize dictionary to store plotting and other parameters
```

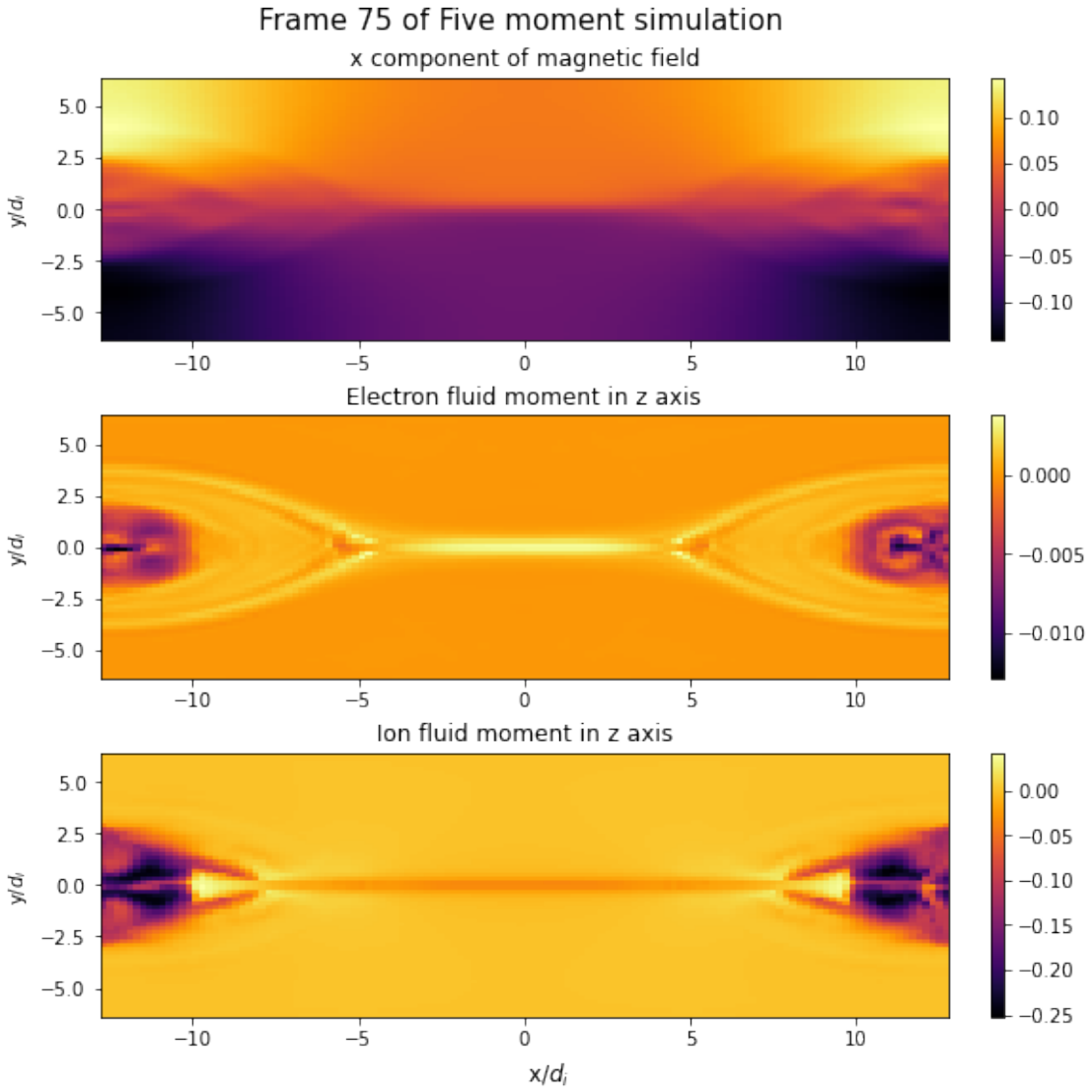
```
# Requires a _params.txt file in your data directory of the form␣
 ↪gkeyllOutputBasename_params.txt! See example_params.txt for formatting

paramsFile = '/home/kasaiyuki/gkylnotebooks/5-moment/rt-5m-gem-params.txt'
suffix = '.gkyl'
varids = ['bx', 'n_elc', 'uz_elc', 'n_ion', 'uz_ion']
nFrame = 100 # Number of frames used in the generating file - corresponds to␣
 ↪number of output file sets made
fileNum = 75
dpsi = [0] * nFrame # Created "empty" list
data = []
```

```
/home/kasaiyuki/pgkylFrontEnd
```

[12]:
```
%%capture
# discards output for this cell
# Read files for the magnetic field
for i in range(len(varids)):
    varid = varids[i]
    var = gkData.gkData(paramsFile,fileNum,suffix,varid,params)
    var.readData()
    data.append(var.data)
    coords = var.coords
    print(np.shape(data))
```

[13]:
```
fig, axes = plt.subplots(nrows=3, figsize=(8, 8), constrained_layout=True,␣
 ↪sharey=True)
fig.supxlabel('x/$d_i$')
fig.suptitle(f"Frame {fileNum} of Five moment simulation", fontsize=15, x=0.475)

# Plotting x component of magnetic field
im1 = axes[0].pcolormesh(coords[0], coords[1], np.transpose(data[0]),␣
 ↪cmap='inferno')
axes[0].set_ylabel("y/$d_i$")
axes[0].set_title("x component of magnetic field")
cbar = fig.colorbar(im1, ax=axes[0])

# Plotting electron fluid moment in z axis
im2 = axes[1].pcolormesh(coords[0], coords[1], np.transpose(data[1]*data[2]),␣
 ↪cmap='inferno')
axes[1].set_ylabel("y/$d_i$")
axes[1].set_title("Electron fluid moment in z axis")
cbar = fig.colorbar(im2, ax=axes[1])

# Plotting ion fluid moment in z axis
mass = 25 # multiply by mass when plotting
```

```
im3 = axes[2].pcolormesh(coords[0], coords[1], np.
  ↪transpose(data[3]*data[4]*mass), cmap='inferno')
axes[2].set_ylabel("y/$d_i$")
axes[2].set_title("Ion fluid moment in z axis")
cbar = fig.colorbar(im3, ax=axes[2])

plt.show()
```



[14]:
```
fig.savefig(f'Frame{fileNum}.png', transparent=True)
```

Note that it took us much longer to get to a similar looking plot, but we also have much more power over size, colors, titles, etc. Also, editing the code to plot different frames is as easy as changing the value of the 'nFrame' variable. Now, let's make a few animations to more easily show the change

over time. First, we need to install ffmpeg with `bash sudo apt install ffmpeg`, then we can use pgkyl to make the movies. Run the commands below and edit the 'img src' in the next cell to view them in the notebook. Sample videos are located in the assets folder to compare with. Don't worry if the commands take a minute or two to finish.

[ ]:
```
%cd /home/kasaiyuki/gkylnotebooks/5-moment
!pgkyl "rt-5m-gem-field_*.gkyl" select -c 3 anim --saveas 'field-movie.gif'
!pgkyl "rt-5m-gem-elc_*.gkyl" select -c 3 anim --saveas 'elc-movie.gif'
!pgkyl "rt-5m-gem-ion_*.gkyl" select -c 3 anim --saveas 'ion-movie.gif'
```

rt-5m-gem-field

rt-5m-gem-elc

rt-5m-gem-ion

6.1 Parameters

In order to more deeply understand the computation, some discussion of the parameters of the simulation is warranted. In the lua file we used to generate the simulation, there are a few parameters that we can change that affect the simulation greatly. The `tEnd` variable changes when the simulation ends as a factor of inverse cyclotron frequency. Currently it is set to run for long enough to see turbulence in the plasma grow. Try increasing it and run the simulation again to see what happens after. The `Nx, Ny` variable governs how many cells are in the simulation. The current value is large enough for the proper physics to occur but not too large that it is difficult for some lower powered machines to run. If you have a capable machine, experiment with changing the grid size and see how long the simulation takes and if there are any noticible changes. The `nFrame` parameter tells the simulation how many frames/file sets to output. If you want more granular details about the simulation you can increase it to see the more minute changes. Other parameters such as `B0` affect more of the physics and may completely change how the simulation runs. If you intend on changing them, make sure to keep a backup of the default values.

Now that we have gone over an example, lets move on to plotting reconnecting fluxes in the `fluxPlotNotebook.ipynb` file.

7 Troubleshooting

- https://stackoverflow.com/questions/66744219/jupyter-lab-networkerror-running-on-wsl2
- If when typing xterm, you get an error similar to `xterm: Xt error: Can't open display: 192.168.128.1:0`, replace the `export display` line in the .bashrc (or equivalent) file with `export DISPLAY=$(route.exe print | grep 0.0.0.0 | head -1 | awk '{print $4}'):0.0`
- If there seems to be no solution to any bugs and errors, try installing an older version of Ubuntu. Check your version by running `lsb_release -a` in the terminal and install an older LTS version from the Microsoft Store.