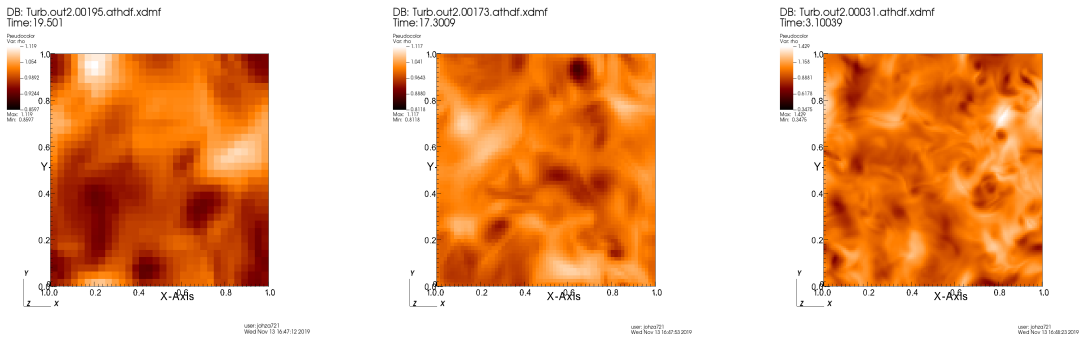
Turbulence in Plasma

Week 1: 11 Nov - 13 Nov

Started this week learning about how to use MPI in Athena++. Followed the 3D blast wave tutorial and managed to get it working on Thunderbird with HDF5. Jono also recommended learning a text editor, so I spent an hour learning the basics of Vim. **Note:** HDF5 is better to use than .vtk with parallel computing as it allows all the processors being used to write the data in one file compared to a file for each processor that need to be joined (less hassle).

Jono then gave me an Athena++ hydrodynamic turbulence input script to play around with, and some MATLAB scripts that analyse the energy spectrum of the fluid in question. At the moment, we model the fluid in a cube. There are two different modes that we're wanting to focus on: decaying turbulence (disturb the fluid initially then leave it to its own devices) and continuously driven turbulence. Ran the Athena++ code with 3 different grid sizes for both modes; see screenshots below.



(a) Grid Size: 32 (b) Grid Size: 64 (c) Grid Size: 128

Figure 1: Face-on view of the 3D forced turbulence simulations with different grid sizes; density plotted

These were just the simulations with none of the parameters changed in the input script. Using the MATLAB scripts I also obtained the energy spectrum of the fluid; was very rough due to the low resolution. Wanted to try start a 256^3 grid size simulation before I left on Wednesday but didn't have enough time to set up; will try again later. Thursday and Friday of this week were spent at the Otago Software Carpentry Workshop.

Next Week: Play around with parameters. Want to run the larger grid size simulations to obtain a better energy spectrum that fits the $k^{-5/3}$ law, will add screenshots of spectrum then. Plot the total energy over time for both modes; should observe fluctuations in the energy for forced turbulence.

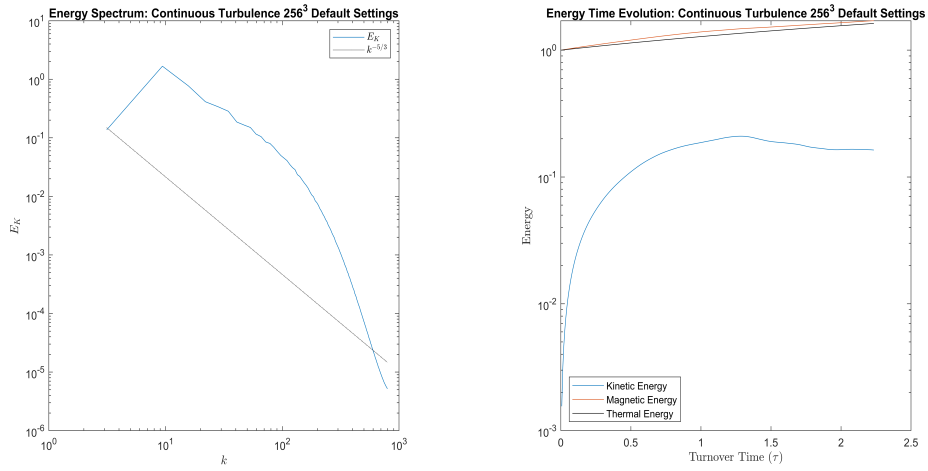
Week 2: 18 Nov - 22 Nov

This week I ran bigger simulations for both decay and forced turbulence from last week in order to be able to plot the energy spectrum and time evolution. The grid size ranged from 32 to 256, and runs over 30 seconds. All simulations left the parameters in `athinput.turb` unchanged.

Calculated the turnover time $\tau \sim L/u_l$ of eddies on the scale of the box to get an idea of the timescales involved in the energy cascade. This is important in the decaying case as all the input energy dissipates within a few turnover times, so this allowed me to get an approximate time range to average the energy spectrum over. For the decay simulations I used $L = 1$ and $u_L = \sqrt{u_x^2 + u_y^2 + u_z^2}$ taken at the start of the simulation; for the continuous simulations the average turnover time in the saturated state was used. The data was gotten from the `Turb.hst` file.

Continuous Forcing: For the continously forced case, the 256 grid size gave the best result. This is expected as it is able to simulate smaller scale eddies compared to lower resolution simulations, allowing more of the energy cascade to be observed. We see that the energy spectrum does follow the $k^{-5/3}$ law (shown in Fig. 2a) for a given range of wavevectors.

The energy evolution (Fig. 2b) shows an increase in kinetic energy until it levels out after a few turnover times. This leveling out is due to the energy dissipation rate matching the energy input rate from the forcing. There is still some variation around the mean value, which arises from fluctuations due to turbulence.



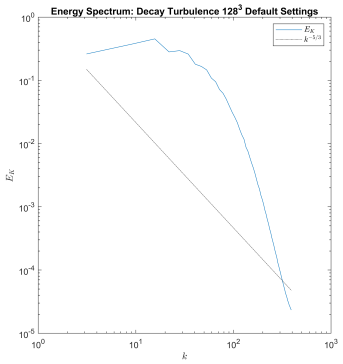
(a) Averaged Spectrum (b) Energy Evolution

Figure 2: Plots showing the energy spectrum and time evolution of the 256 grid size continuous forcing simulation

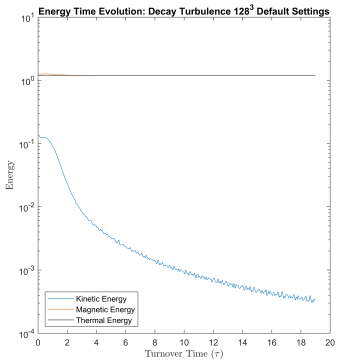
Decay: The energy spectrum (Fig 3a), averaged over the first two turnover cycles, is not

as well defined as the continuous case. I think this is because the energy depends on time instead of being approximately constant. The energy evolution (Fig. 3b) shows that the total kinetic energy decreases over time due to dissipation from viscosity.

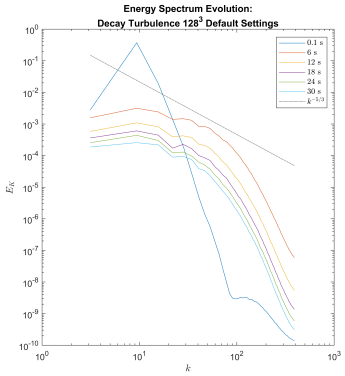
For the spectrum evolution (Figure 3c) I took snapshots of the spectrum at 3 second intervals, with a snapshot at 0.1 seconds to observe the energy input (the curve sharply peaked at $k \sim 10$). We see that the energy cascades down through the length scales directly after input (at 6 seconds), and then decreases as it is dissipated as heat at the micro scale (shown by the "sinking" of the spectrum). This is expected as we saw that the total energy in Fig. 3b is decreasing.



(a) Averaged Spectrum



(b) Energy Evolution



(c) Spectrum Evolution

Figure 3: Plots showing the energy spectrum and time evolution of the 128 grid size decay simulation

The magnetic and thermal energy had no relevance to these simulations; it's just part of the MATLAB script that I forgot to remove.

I learnt that it's good to run the simulations at different resolutions starting with the lowest as it allows you to get a rough idea of what is going to happen without the expense of computing time. It also helps as you can compare with the model to see whether the configuration used is worth investigating. The higher resolutions could differ as turbulence depends strongly on all length scales in the inertial subrange, which are included in the bigger simulations, but it still helps to see what could happen.

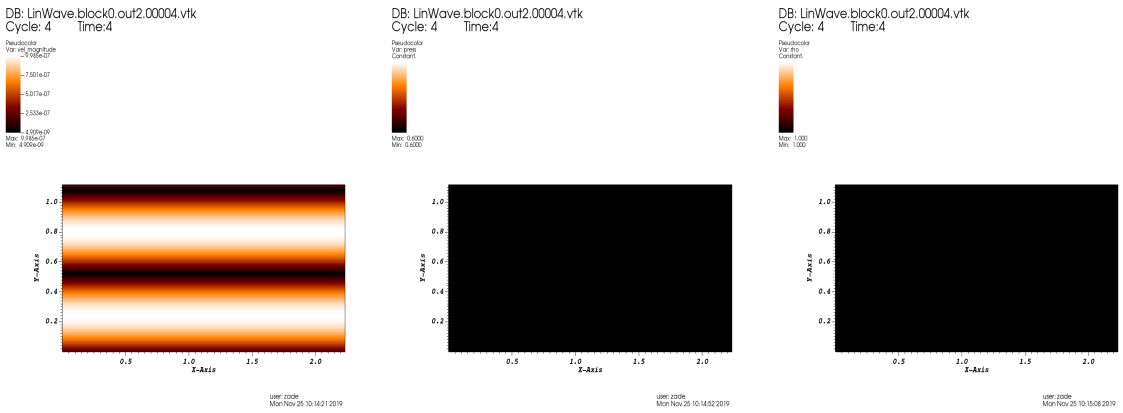
Week 3: 25 Nov - 29 Nov

Ran the linear wave problem in Athena to observe Alfv́en waves. The pressure and density of the plasma should be constant as these waves travel, which was observed in the simulation (Fig. 4). Alfv́en waves have the following dispersion relationship:

$$\omega = k_{\parallel} v_A$$

where $k_{\parallel}$ is the component of the wavevector parallel to the magnetic field and $v_A = B_0/\sqrt{4\pi\rho}$ is the Alfv́en velocity at which the waves move, where B_0 and ρ are the magnetic field strength and fluid density.

The linear wave problem has the default settings (I'm not sure about units): $\rho = 1$, $\mathbf{B}_0 = (1, \sqrt{2}, 0.5) \implies B_0 = \sqrt{13}/2$. This gives a value for the Alfv́en velocity of $v_A \approx 0.5086$. The next thing to do is to see whether the wave gives us this value through the dispersion relationship above.



(a) Velocity Magnitude (b) Pressure (c) Density

Figure 4: Plots of different properties of the fluid as a linear wave passes through

I wrote a small Python script to calculate and compare the simulation and theoretical Alfv́en velocities. The angular frequency was calculated by measuring the (simulated) time it took for a peak of the wave to travel through the box, as it has periodic boundary conditions, and dividing this by 2π. The magnitude of the wavevector was gotten by dividing 2π by the wavelength. All quantities were eyeballed as only a quick check was needed. The result is seen in Fig. 5, showing the dispersion relationship holds in the simulation.

```
print("Alfven speed:", v_A) Alfven speed: 0.5085536181410275
print("Calculated speed:", v_pred) Calculated speed: 0.5045045045045046
print("Ratio:", v_pred / v_A) Ratio: 0.9920379808695018
```

Figure 5: Output of the Python script I wrote

Questions (answered):

- Is **k** automatically parallel to **B** when the `ang_3_vert` variable is set to `True`?

  No, it just rotates the axes such that **k** is along the y-axis. Can make it parallel to **B** through code.

- Why are the density and pressure constant for an Alfv́en wave?

  Because the Alfv́en wave is a **shear wave** i.e. no compression at all. This means $\nabla \cdot \mathbf{u} = 0 \implies D\rho/Dt = 0$ i.e. the density is constant in time and thus so is the pressure.

- Why is a rectangle used insted of a square for the region of interest?

  Just to make the numbers work out right such that the waves are continuous over the periodic boundaries. Can make it a square if needed.

The next step was simulating turbulence in the case of MHD. Essentialy the same code was used as the hydrodynamic turbulence case, with a couple of updates to allow for the initialisation of a mean magnetic field in the region of interest.

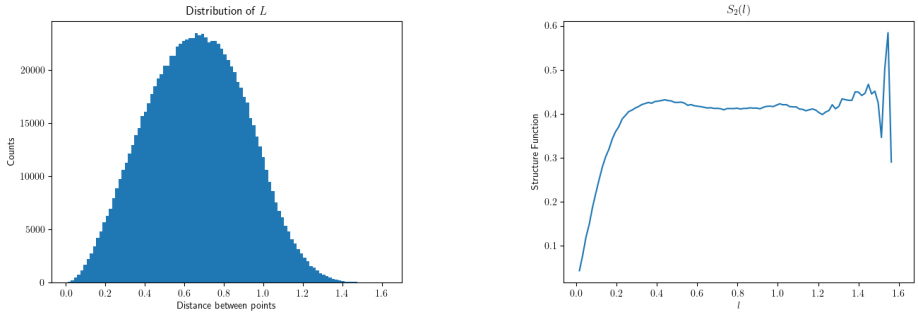
In order to get critical balance, we want $u_\perp \sim v_{A0} = B_0/\sqrt{4\pi\rho}$. Athena sets $4\pi \to 1$, so we get $v_{A0} = B_0/\sqrt{\rho}$. We can set $B_0 = 1$ and $\rho = 1$ such that $u_\perp \sim v_{A0} = 1$. To do this in the simulation I've set $dE/dt = 1$ as this means $u_\perp^3/L \sim 1$ (the box size has $L = 1$; letting everything work out to 1 makes it simpler).

Forgot to add the magnetic field flag so have to redo larger simulations. **A good check is to look at the first few snapshots in VisIt to see if the magnetic fields (or other parameters) are set up the way I wanted.**

Trying out Athena's Python codes for plotting and reading data files to help write the `spectrum.m` code in Python. Haven't made much progress yet as I've been focusing on writing the `structure_function.py` code.

Originally wrote `structure_function.py` using just lists. For 1 million random pairs of points on a 128^3 continuously forced turbulence grid (non-MHD), the average run time was about 100 seconds. After learning about `numpy` and utilizing this, the same problem had its run time reduced to about 20 seconds; this is a massive improvement. Not sure whether I need to test for a larger number of points. Below in Fig. 6 I've added the inital output of the function; showing the distribution of distances between the points and the velocity structure function at $t = 0.1$ s.

I've just realised that it doesn't look right in Fig. 6 as I've used the snapshot in the first 0.1 second after the simulation has begun, so the turbulence will not have fully developed. Will look into more detail next week.



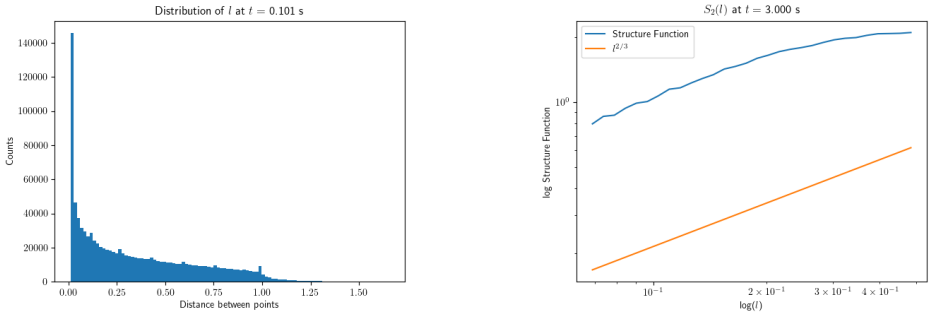
(a) Distribution of lengths in box (b) Velocity Structure Function

Figure 6: Output of the structure function code for the hydrodynamic 128^3 forced turbulence simulation

Week 4: 2 Dec - 6 Dec

After talking with Jono about the structure function code I wrote last week, I updated it such that it generates a log spaced l-grid and generates the random points closer together in order to be able to see the finer details in the small l range (inertial subrange). After updating the code we l distribution and structure functions are shown in Fig. 7. The lengths between the points are now skewed to be smaller which is what was wanted, and the structure function is very close to following Kolmogorov's $l^{2/3}$ rule.

I've created a GitHub repo to hold the Python code that calculates these here.



(a) Distribution of lengths in box after updating code (b) Log plot of velocity structure function after updating code

Figure 7: Output of the structure function code for the hydrodynamic 128^3 forced turbulence simulation

Updated the code to calculate the $l_\parallel$ and $l_\perp$ components of the separation vector relative to the magnetic field to observe the critical balance law after a struggle getting it to work.

Next step is to talk with Jono about working with his modified Athena code.