CTA200 Project

Shashvat Varma

May 2024

Question 1

In this section, we create a more efficient algorithm to compute eigenvalues and eigenvectors of Hermitian matrices. To do so, we encorporate both parallelization across multiple cores on the computing system, as well as vectorize the calculation to speed it up. The algorithm was then used to find the eigenvalues of the symmetric component of the velocity gradient tensor in a magnetohydrodynamic simulation. The code generates an animation of how the eigenvalues evolve in time. An example frame from the animation is shown in Figure 1. The function developed can be used on any individual matrix.

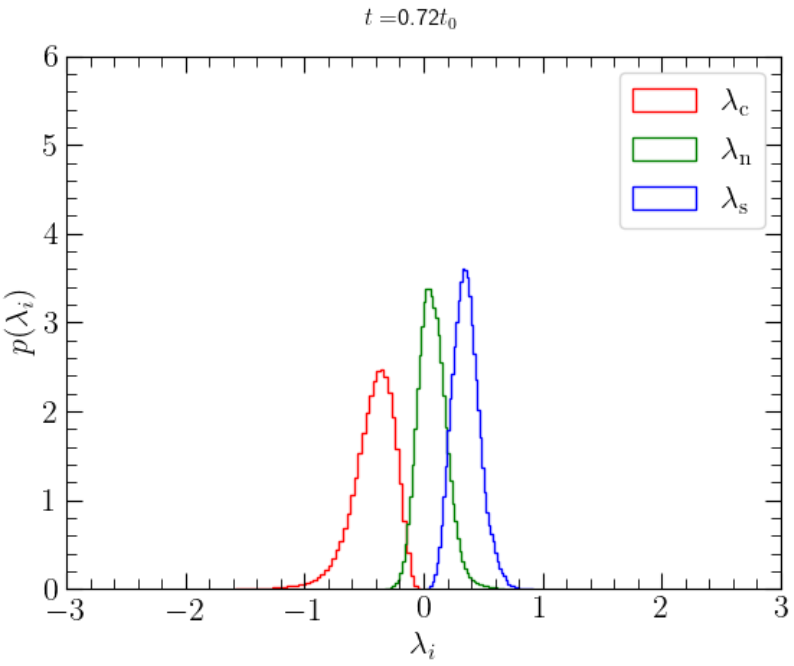

Figure 1: Histogram of the eigenvalue spectrum, found by computing the eigenvalues of the symmetric component of the velocity gradient tensor at each of the 128^3 points in the simulation. This particular image is the distribution at time $0.72t_0$.

Question 2

In this section, we make use of the closed-form solutions to the eigenvalue problem outlined by Deledalle et al. 2017. Consider a Hermitian matrix of the form:

$$A = \begin{pmatrix} a & d^* & f^* \\ d & b & e^* \\ f & e & c \end{pmatrix} \tag{1}$$

The eigenvalues λ_i of the matrix A is given by:

$$\lambda_1 = [a + b + c - 2\sqrt{x_1}\cos(\phi/3)]/3 \tag{2}$$
$$\lambda_2 = [a + b + c + 2\sqrt{x_1}\cos[(\phi - \pi)/3]]/3 \tag{3}$$
$$\lambda_3 = [a + b + c + 2\sqrt{x_1}\cos[(\phi + \pi)/3]]/3 \tag{4}$$

where we define

$$x_1 = a^2 + b^2 + c^2 - ab - ac - bc + 3(|d|^2 + |f|^2 + |e|^2) \tag{5}$$
$$x_2 = -(2a - b - c)(2b - a - c)(2c - a - b) + 9\left[(2c - a - b)|d|^2 + (2b - a - c)|f|^2 + (2a - b - c)|e^2|\right] - 54\mathrm{Re}(d^* e^* f) \tag{6}$$

$$\phi = \begin{cases} \arctan\left(\frac{\sqrt{4x_1^3 - x_2^2}}{x_2}\right), & x_2 > 0 \\ \pi/2, & x_2 = 0 \\ \arctan\left(\frac{\sqrt{4x_1^3 - x_2^2}}{x_2}\right) + \pi, & x_2 < 0 \end{cases} \tag{7}$$

With these eigenvalues, we can define the eigenvectors v_i as:

$$v_1 = \begin{pmatrix} (\lambda_1 - c - em_1)/f \\ m_1 \\ 1 \end{pmatrix} \tag{8}$$
$$v_2 = \begin{pmatrix} (\lambda_2 - c - em_2)/f \\ m_2 \\ 1 \end{pmatrix} \tag{9}$$
$$v_3 = \begin{pmatrix} (\lambda_3 - c - em_3)/f \\ m_3 \\ 1 \end{pmatrix} \tag{10}$$

where

$$m_1 = \frac{d(c - \lambda_1) - e^* f}{f(b - \lambda_1) - de} \tag{11}$$
$$m_2 = \frac{d(c - \lambda_2) - e^* f}{f(b - \lambda_2) - de} \tag{12}$$
$$m_1 = \frac{d(c - \lambda_3) - e^* f}{f(b - \lambda_3) - de} \tag{13}$$

This method for computing eigenvalues was implemented in Python to compute eigenvalues of the symmetric component of the velocity gradient tensor in a magnetohydrodynamic simulation. The code generates an animation of how the eigenvalues evolve in time.

Traditional eigenvalue solvers like ones found in the numpy and scipy libraries use root solvers to find the roots of the characteristic polynomial of the matrix. This process is significantly slower than direct computations. In the code file, I time the computation of 128^3 eigenvalues. The results are summarized in Table 1.

Method	Computation Time (s)
numpy.linalg.eigh	24
scipy.linalg.eig	70
Deledalle et al. 2017	1.5

Table 1: Time taken to compute eigenvalues of 128^3 symmetric matrices using different numerical methods. Direct computaiton is clearly faster than traditional methods through solving the characteristic polynomial.

Another reason for this significant time difference is in how the three routines were implemented. The structure of the arrays were of shape (3,3,128,128,128), and vectorized calculations from the first two methods

would require an array of (128,128,128,3,3). There is no simple reshape function that converts between the two without scrambling the values in each matrix. To construct this array, we would have to do element-wise redefinition anyway. So, the numpy and scipy calculations were done by looping through each individual matrix. The beauty of the third method is that direct computations can be done with any numpy array shape, effectively vectorizing the problem and speeding up the calculation. Both methods yield identical results up to numerical precision, as shown in Figures 2 and 3.

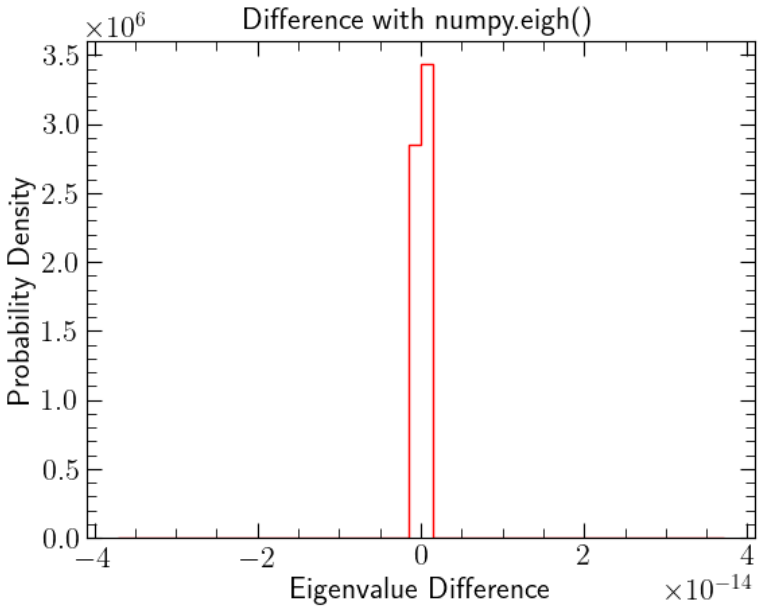

Figure 2: The distribution of difference in eigenvalues computed by numpy and computed by the new numerical method at a single time realization.

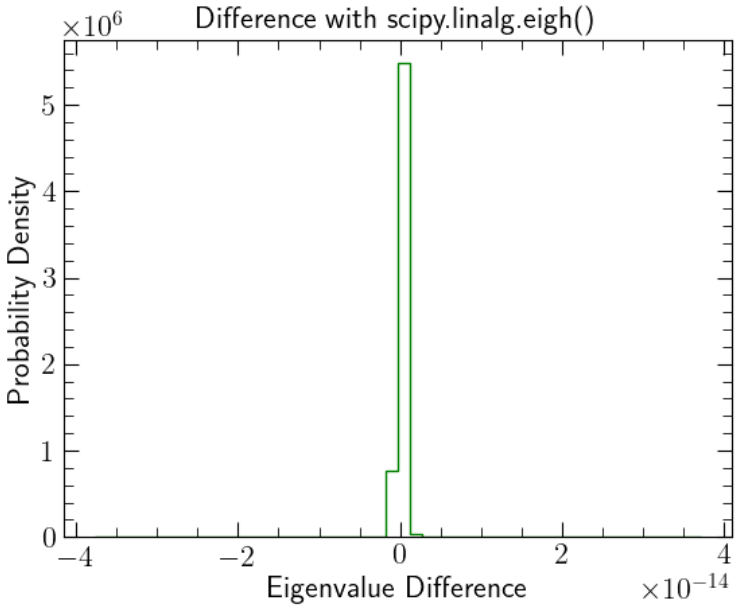

Figure 3: The distribution of difference in eigenvalues computed by scipy and computed by the new numerical method at a single time realization.

Question 3

In this section, we additionally compute the angle between the magnetic field and the eigenvector at each point in the simulation. This is done through computing the dot product. The results for the two different phases of the dynamo are shown in Figures 4 and 5.

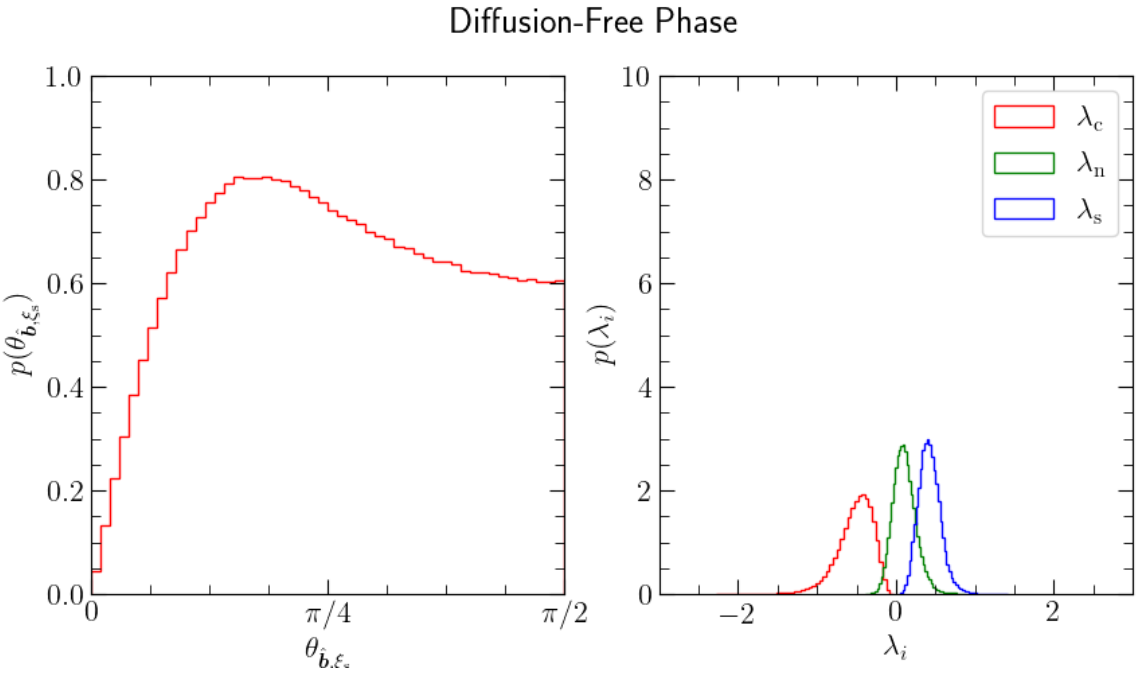

Figure 4: The angle between the magnetic field and the stretching eigenvector associated with eigenvalue λ_s during the diffusion-free phase.

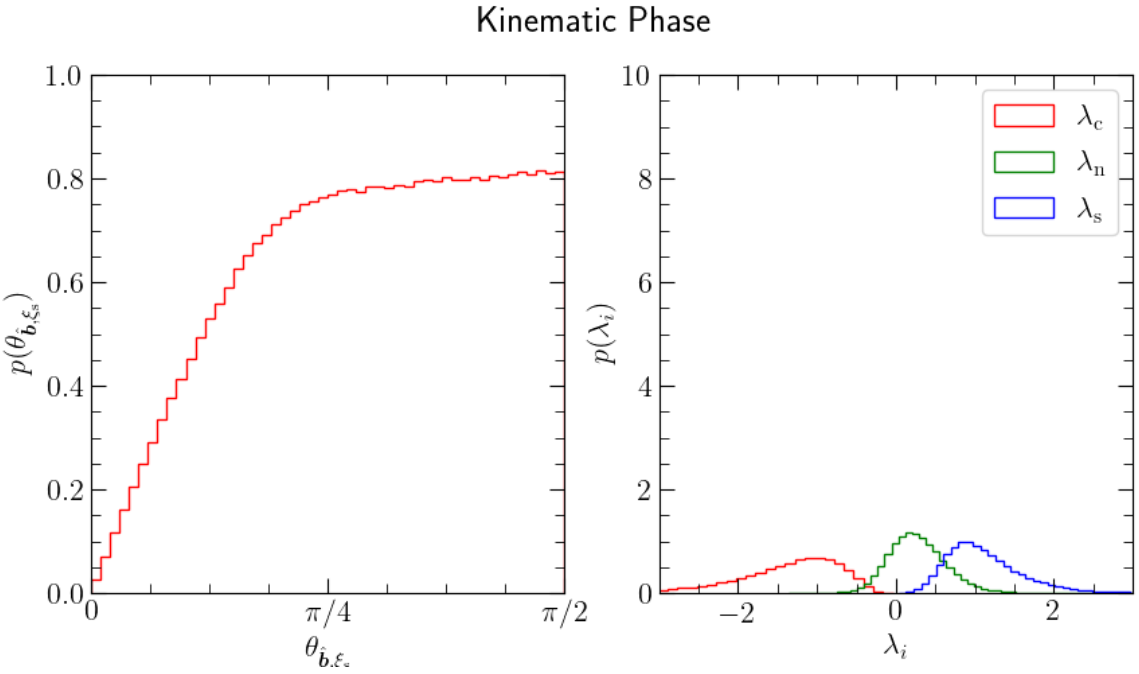

Figure 5: The angle between the magnetic field and the stretching eigenvector associated with eigenvalue λ_s during the kinematic phase.

It is evident that during the diffusion-free phase, the stretching eigenvector prefers to align parallel/antiparallel with the magnetic field, while it prefers to be mostly perpendicular during the kinematic phase.