# 3b Magnetohydrodynamics (MHD)

Johannes Winther, Sebastian Utecht, Jakob Schauser

December 10, 2023

## 1 Solving the MHD equations using numerical tools - Introduction to FARGO3D

### 1.1 Task 1: For the Orszag-Tang Vortex test, plot $B_y$, $B_z$, $V_y$ and $V_z$ for $t = 0.5$

Below we see the subplots of $B_y$, $B_z$, $V_y$ and $V_z$ for $t = 0.5$:

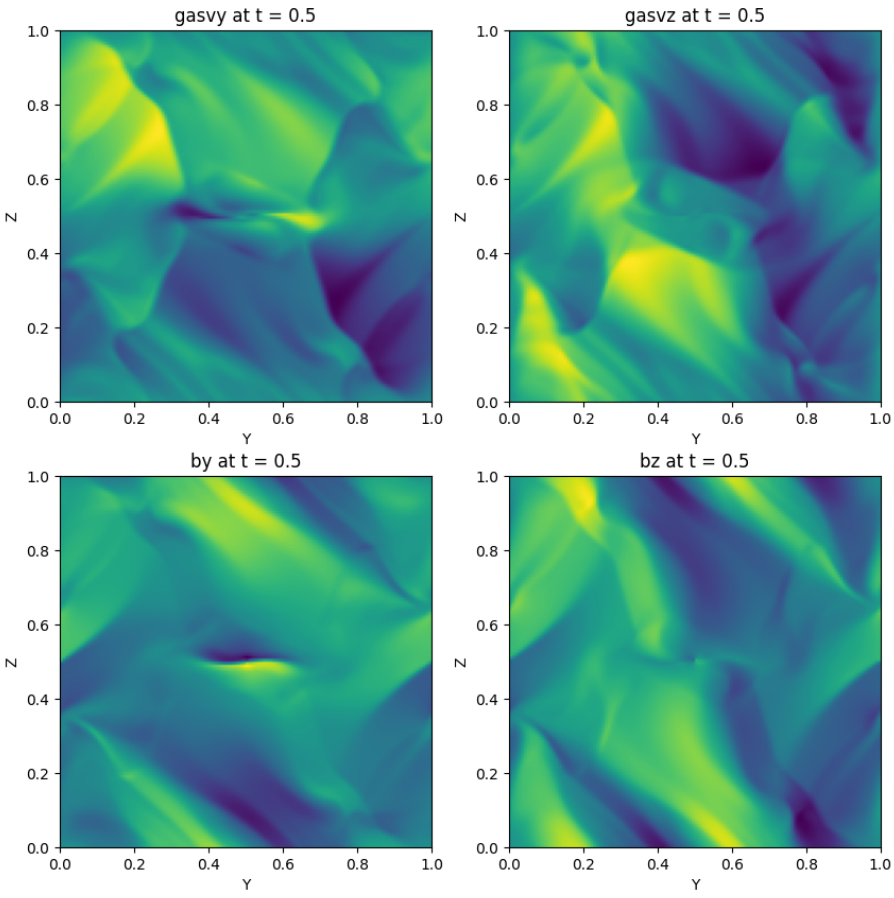


Figure 1:

## 1.2 Task 2: For the Orszag-Tang Vortex test, plot the streamlines and the magnetic lines

Below on the left we see the field lines of the B field, and on the right we see the field lines of the velocity field.

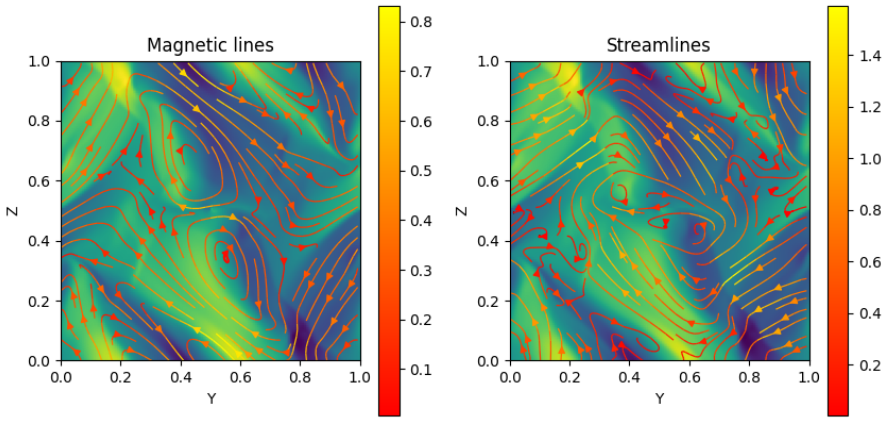


Figure 2:

## 1.3 Task 3: Calculate $\nabla \cdot \mathbf{B}$, plot it with imshow and verify $\nabla \cdot \mathbf{B} = 0$ for all the mesh. Consider the solution at $t = 0.5$

Below is a code section which shows how the divergence is computed by the left slope derivative.

```
#get data
n  = 10
by = get_data(names[2],n,params)
bz = get_data(names[3],n,params)

#Computing left slope and using as derivative
divB = (by - np.roll(by,-1,axis=1))+(bz - np.roll(bz,-1,axis=0))

#Plotting
div=plt.imshow(divB)
plt.colorbar(div)
plt.title("Divergence of B at t = 0.5")
plt.xlabel("Y")
plt.ylabel("Z")

print("RMS",np.sqrt(np.mean(divB**2)))
```

On the plot below, and using the colorbar, we can see that the divergence is essentially zero as expected. Also when calculating RMS of the divergence we get $RMS(\nabla \cdot B) \approx 4.43 \cdot 10^{(-16)} \approx 0$.

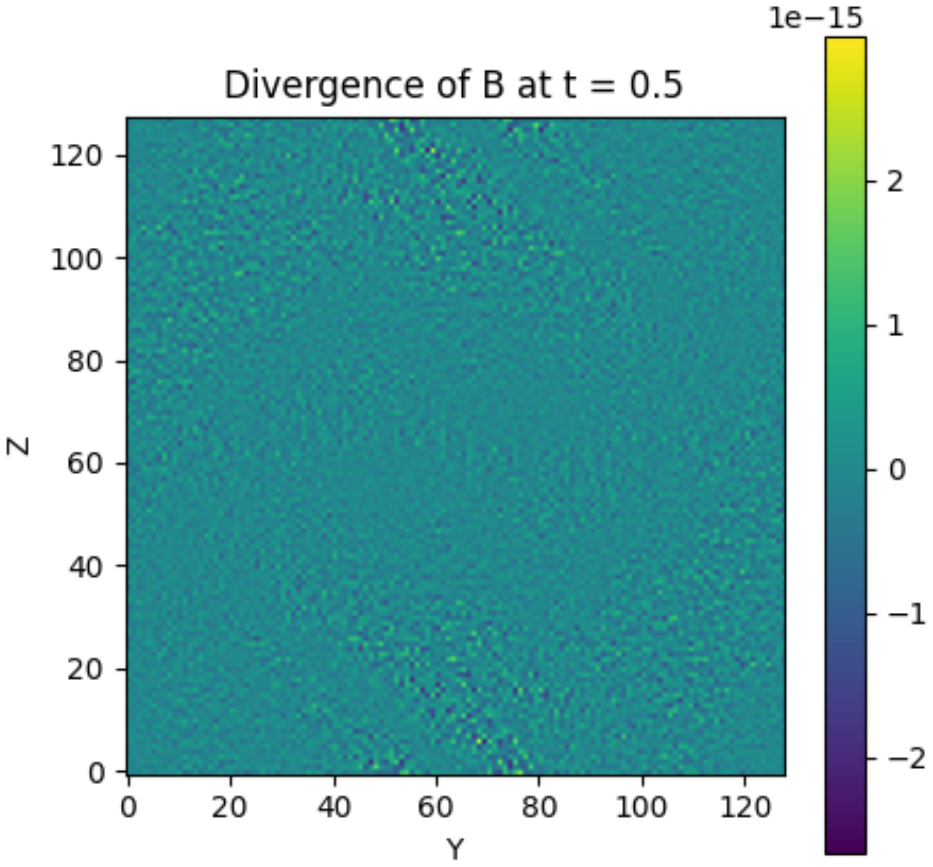


Figure 3:

# 2 Linear MRI test

## 2.1 Task 1

Below I find the maximum by setting the derivative equal to zero and isolating:

$$\begin{aligned}
\omega_{\text{unstable}} &= \left[-\frac{1+2\tilde{k}^2}{2}+\frac{1}{2}\sqrt{1+16\tilde{k}^2}\right]^{1/2} \\
\Rightarrow {\omega_{\text{unstable}}}^2 &= \left[-\frac{1+2\tilde{k}^2}{2}+\frac{1}{2}\sqrt{1+16\tilde{k}^2}\right] \\
\Rightarrow 0 = \frac{\mathrm{d}}{\mathrm{d}\tilde{k}}{\omega_{\text{unstable}}}^2 &= -2\tilde{k}+8\tilde{k}\frac{1}{\sqrt{1+16\tilde{k}^2}} \\
\Rightarrow 1 &= 4\frac{1}{\sqrt{1+16\tilde{k}^2}} \\
\Rightarrow \tilde{k}_{max} &= \sqrt{15}/4
\end{aligned} \tag{1}$$

Below I insert $\tilde{k}_{max}$ and get $\omega_{max}$ :

$$\begin{aligned}\omega_{max}(\tilde{k_{max}}) &= \left[-\frac{1+2\frac{15}{16}}{2}+\frac{1}{2}\sqrt{1+16\frac{15}{16}}\right]^{1/2} \\ &= \left[-\frac{1}{2}-\frac{15}{16}+2\right]^{1/2} \\ &= \frac{3}{4}\end{aligned} \tag{2}$$

Below we can see that the LHS of the middle equation grows with power 2, compared to the RHS which grows with power 1, which means that LHS becomes larger than RHS for $\tilde{k} > \sqrt{3}$ and we then get complex $\omega$ because the negative part is larger than the positive inside the square root as can be seen in eq. 4.

$$0 = \omega \Rightarrow 1 + 2\tilde{k}^2 = \sqrt{1+16\tilde{k}^2} \Rightarrow \tilde{k} = \sqrt{3}, 0 \tag{3}$$

$$\Rightarrow \omega_{\text{unstable}}(\tilde{k} > \sqrt{3}) = \left[-\frac{1+2\tilde{k}^2}{2}+\frac{1}{2}\sqrt{1+16\tilde{k}^2}\right]^{1/2} = \text{complex}, \quad \text{since} \quad \frac{1+2\tilde{k}^2}{2} > 2\sqrt{1+16\tilde{k}^2} \tag{4}$$

Inserting the found values, we get the plot below (fig 4) which is as expected.

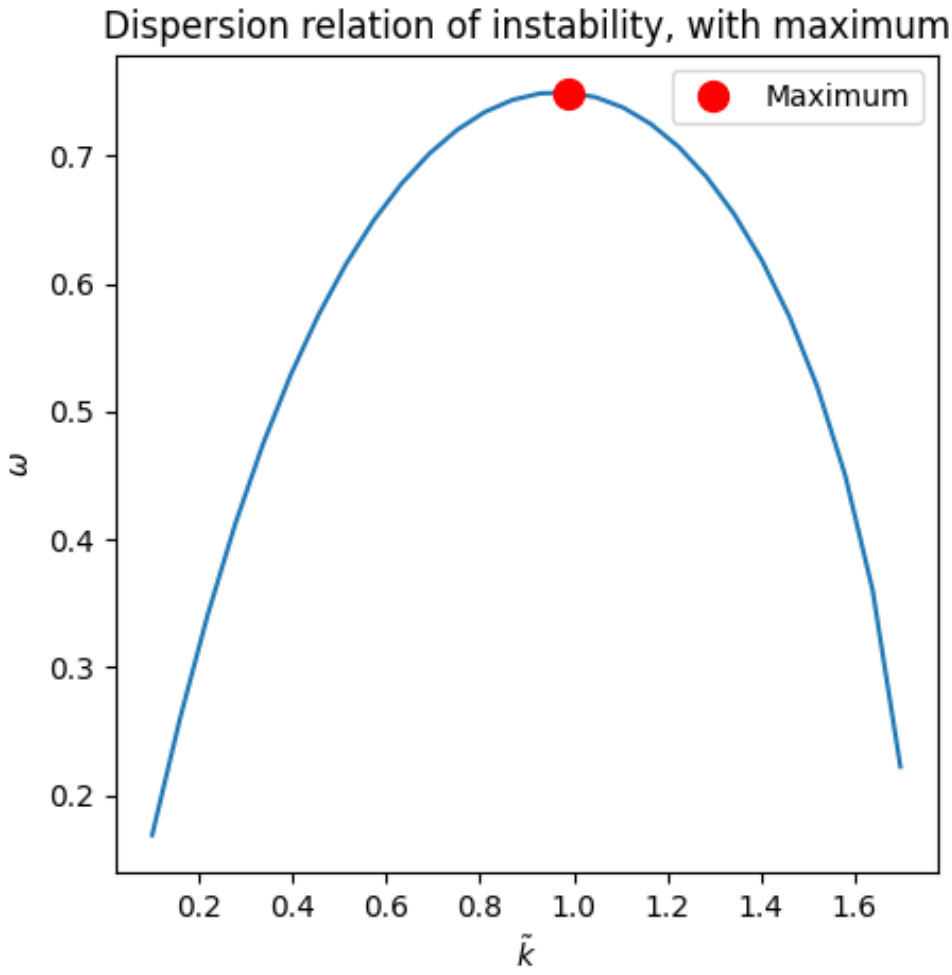


Figure 4:

### 2.2 Task 2 - Using FARGO3D, measure the linear growth rate $\omega$ of the MRI for different values of $\tilde{k}$

After setting up and running the code with the new initial conditions we plot Bx($z$,$y_0$) for different n, as shown below on fig 5.

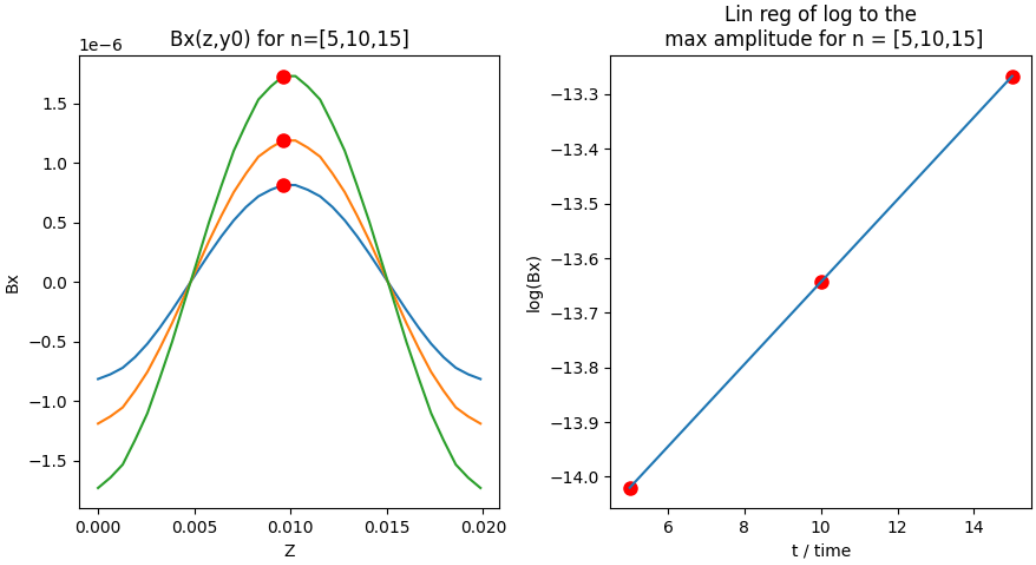


Figure 5:

Code used for the above is:

```
params = Parameters("outputs/mri-growth-sb/")
plt.figure(figsize=(9,9))

# First, load the 1D arrays with the domain
y_1d  = np.loadtxt(params.outputdir+"domain_y.dat")[3:-4]
z_1d  = np.loadtxt(params.outputdir+"domain_z.dat")[3:-4]

# Now, create a 2D mesh using meshgrid
y_2d,z_2d = np.meshgrid(y_1d,z_1d)

#get data
data1=get_data("bx",5,params)[:,params.ny//2]
data2=get_data("bx",10,params)[:,params.ny//2]
data3=get_data("bx",15,params)[:,params.ny//2]

#z in 1d
z_2d=z_2d[:,params.ny//2]

#find max
i1=np.nanargmax(data1)
```

```
i2=np.nanargmax(data2)
i3=np.nanargmax(data3)

#plot and fit
plt.subplot(2,2,1)
plt.title("Bx(z,y0) for n=[5,10,15]")
plt.xlabel("Z")
plt.ylabel("Bx")

plt.plot(z_2d,data1)
plt.plot(z_2d,data2)
plt.plot(z_2d,data3)
plt.plot(z_2d[i1],data1[i1],"r.",markersize=16)
plt.plot(z_2d[i2],data2[i2],"r.",markersize=16)
plt.plot(z_2d[i3],data3[i3],"r.",markersize=16)

plt.subplot(2,2,2)
plt.title("Lin reg of log to the\n max amplitude for n = [5,10,15]")
plt.xlabel("t / time")
plt.ylabel("log(Bx)")

x=np.array([5,10,15])
y=np.array([data1[i1],data2[i2],data3[i3]])

plt.plot(x,np.log(y),"r.",markersize=16)

slope, intercept = np.polyfit(x, np.log(y), 1)  # 1 represents linear fit
    (degree 1)

# Predicted values using the linear model
y_predicted = slope * x + intercept

plt.plot(x,y_predicted,"-")
plt.tight_layout()
```

On fig 5, we see that the amplitude of the trig-function grows with time, with a faster than linear growth rate. On the right we can see from the linear regression on x,log(y) that it grows exponentially in time, just like expected.

A more detailed regression of this with more points on left of fig 6 gives a slope, which corresponds to $\omega \approx 0.75$, which is still the maximum of the dispersion relation. This is expected because we used $\Delta Z_{max}$ in the simulation:

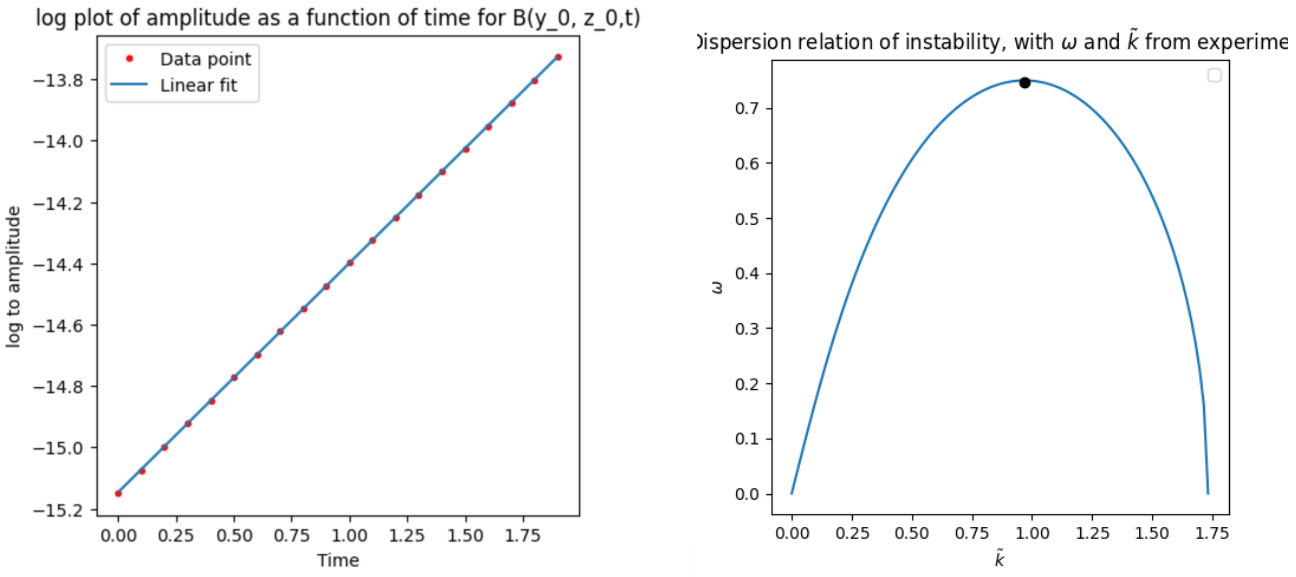


Figure 6:

The code used for the above was:

```
params = Parameters("outputs/mri-growth-sb/")

# We store the data in two lists
bx   = []
time = []

for i in range(20):
    time.append(params.dt*params.ninterm*i)
    data=get_data("bx",i,params)[:,params.ny//2]
    i=10#
    bx.append(data[i])

# We convert the lists to arrays so we can use numpy to modify them (if
    needed)
time = np.array(time)
bx   = np.array(bx)

# Fit
slope, intercept = np.polyfit(time, np.log(bx), 1)  # 1 represents linear
    fit (degree 1)

# Predicted values using the linear model
y_predicted = slope * time + intercept

# Plot
plt.plot(time,np.log(np.abs(bx)),"r.",label="Data point")
plt.plot(time,y_predicted,label="Linear fit")
plt.xlabel("Time")
plt.ylabel("log to amplitude")
plt.title("log plot of amplitude as a function of time for B(y_0, z_0,t)")

plt.legend()

print("slope is", slope)
```

After running simulations with $\Delta Z = [0.0911, 0.0514, 0.0358, 0.0274, 0.0223, 0.0188, 0.0162, 0.0142, 0.0127]$,

we compute the corresponding $\omega = [0.34, 0.52, 0.64, 0.71, 0.75, 0.75, 0.71, 0.62, 0.48]$ in the same way as above, and – analytically by $\tilde{k} = 2\pi v_a/(\Delta Z \Omega_0)$ - the $\tilde{k} = [0.22, 0.39, 0.55, 0.73, 0.89, 1.06, 1.23, 1.4, 1.56]$. Plotting these points, we get the plot shown below on fig 7.

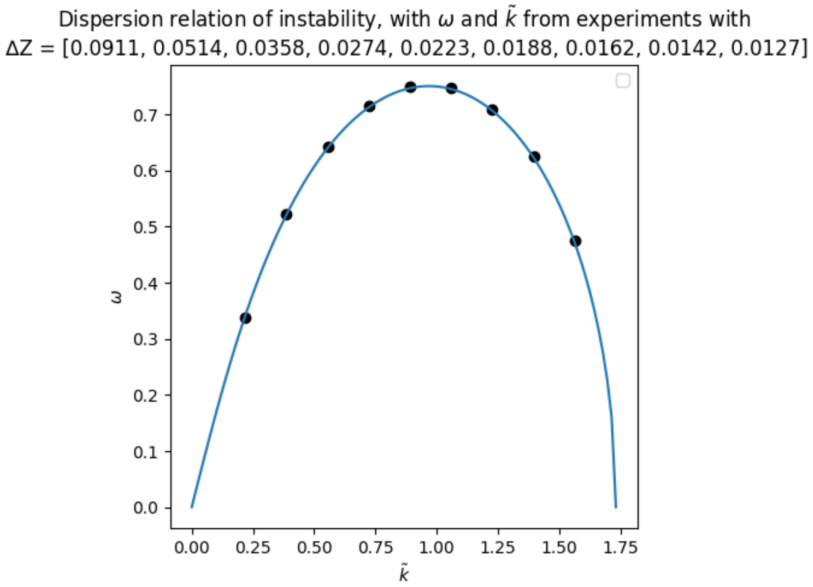


Figure 7:

The code for this is shown below.

```
slopes=[]
anal_k=[]
omega=[]

for i,dz in enumerate(Dz):

    params = Parameters(f"outputs/output{i}/")

    # We store the data in two lists
    bx   = []
    time = []

    #choose some point for each time step
    for i in range(20):
        time.append(params.dt*params.ninterm*i)

        data=get_data("bx",i,params)[:,params.ny//2]
        i=np.nanargmax(data)
        bx.append(data[i])

    #compute analytic
    anal_k.append(0.003162*2*np.pi/dz)

    # We convert the lists to arrays so we can use numpy to modify them (
       if needed)
    time = np.array(time)
    bx   = np.array(bx)

    # Make linreg to get slope
    slope, intercept = np.polyfit(time, np.log(bx), 1)

    slopes.append(slope.copy()
```