Midterm Exam (part 3) - Computational Physics I

Deadline: Friday 25 October 2024 (by 17h00)

12/12

Excellent!

Name: Alan Palma Travez

Part 3. (12 points) Data analysis and visualisation (Orszag-Tang MHD vortex)

The **Orszag-Tang vortex** system describes a doubly periodic ideal gas configuration leading to 2D supersonic magnetohydrodynamical (MHD) turbulence. Although an analytical solution is not known, its simple and reproducible set of initial conditions has made it very appealing for the comparison of MHD numerical solvers. The computational domain is a **periodic box** with dimensions: $[0,2\pi]^2$, and the gas has an adiabatic index $\gamma=\frac{5}{3}$.

In code units, the initial conditions are given by:

$ \vec{v}_{code} = \left(-\sin y,\, \sin x\right) \,,\qquad \vec{B}_{code} = \left(-\sin y,\, \sin 2x\right) \,,\qquad \rho_{code} = 25/9\,,\qquad p_{code} = 5/3, $

and the numerical simulation produces 61 VTK files stored in:

- the **Orszag-Tang-MHD** folder:

https://github.com/wbandabarragan/physics-teaching-data/blob/main/2D-data/Orszag_Tang-MHD.zip

which also contains the following descriptor files:

- a **units.out** file that contains the CGS normalisation values.
- a **vtk.out** file whose second column contains the times in code units.
- a **grid.out** file that contains information on the grid structure.

You can use VisIt to inspect the data. The written fields are:

- density (rho)
- thermal pressure (prs)
- velocity_x (vx1)
- velocity_y (vx2)
- magnetic_field_x (Bx1)
- magnetic_field_y (Bx2)

Reference paper: https://arxiv.org/pdf/1001.2832.pdf

"High-order conservative finite difference GLM-MHD schemes for cell-centered MHD", Mignone, Tzeferacos & Bodo, JCP (2010) 229, 5896.

Tasks:

Within a single python notebook, carry out the following tasks:

(a) Create a python function that reads the **units.out** file, stores the normalisation values for length, velocity, and density, calculates the normalisation values for thermal pressure, magnetic

field, and time, and returns them all into tuple objects.

Note: As shown in class, the normalisation values for thermal pressure ($p_0=\rho_0\,v_0^2$), magnetic field ($B_0=\sqrt{4\,\pi\,\rho_0\,v_0^2}$), and time ($t_0=\frac{L_0}{v_0}$) can be derived from the length, velocity, and density values.

(b) Create a python function that reads the **vtk.out** file, reads the second column, and returns the times in CGS units using t_0 from point (a).

(c) Create a python function that reads a VTK data file, normalises the data fields to CGS units using the values from points (a) and (b), and returns them jointly with the mesh and time information as tuple objects.

(dx2) Call all the above functions for VTK file # 30 of each simulation, and make the following maps using the correct mesh coordinates, dimensions and time, all in in CGS units:

- A figure showing the gas density, ρ.
- A figure showing the gas sound speed, $c_s=\sqrt{\gamma\frac{p}{\rho}}$.
- A figure showing showing the kinetic energy density, $E_k = \frac{1}{2}\rho\,v^2$ with $v=\sqrt{v_x^2+v_y^2}$.
- A figure showing the magnetic vector field, $\vec{B}= \vec{B}_x + \vec{B}_y$.

Notes: Choose different perceptually-uniform colour schemes for each of the above quantities, and fix the colour bar limits. Add the correct time stamp in CGS units to each map. Since these are high-resolution models, one way to improve the visualisation of 2D vector fields is to interpolate them into a coarser grid.

(e) Create a set of Python functions that loops over all the VTK simulation files and returns maps of the density field ρ, the kinetic energy density E_k, and also histograms of the density field in CGS units for all times, into a folder called "output_data".

(f) Briefly describe: what happens with the density field as time progresses? Does the density field follow any statistical distribution at late times?

(gx2) Create a set of Python functions that loops over all the VTK simulation files, computes the following quantities in CGS units for each time:

- the average gas temperature, $\overline T$, (**Hint:** the temperature in each grid cell can be calculated using the equation of state for ideal gases, i.e., $p=\frac{\rho k_B\,T}{\mu m_u}$, where k_B is the Boltzmann constant, m_u is the atomic mass unit, and $\mu=0.6$ is the mean particle mass in the gas.)
- the average kinetic energy density, $\overline E_k$, where $E_k = \frac{1}{2}\rho\,v^2$.
- the average magnetic energy density, $\overline E_m$, where $E_m = \frac{1}{2}\frac{B^2}{\,\mu_0}$, where $\mu_0\equiv$ magnetic permeability of free space.

and returns:

- a CSV file with 4 columns, time on the first column, and the above quantities in the next ones. The CSV file should be named "stats.csv" saved into the folder called "output_data".

(h) Create a Python function that reads in the CSV file created in (g) and returns (i.e. shows or saves) high-quality labeled figures of each of the above-computed quantities versus time, into

the folder called "output_data".

(i) Briefly describe: Does the flow reach steady state? Which energy density is dominant?

(j) Create a Python function that returns movies showing the time evolution of the kinetic energy density maps computed in (d) and their average values calculated in (g). The movies should be saved into the folder called "output_data".

Solution

(a) Create a python function that reads the **units.out** file, stores the normalisation values for length, velocity, and density, calculates the normalisation values for thermal pressure, magnetic field, and time, and returns them all into tuple objects.

Note: As shown in class, the normalisation values for thermal pressure ($p_0=\rho_0\,v_0^2$), magnetic field ($B_0=\sqrt{4\,\pi\,\rho_0\,v_0^2}$), and time ($t_0=\frac{L_0}{v_0}$) can be derived from the length, velocity, and density values.

In [1]:
```python
#Third party libraries

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pyvista as pv
import scipy.constants as const
import scienceplots
```

In [2]:
```python
#Function
def io_norm_values(path):
    """
    Function to read .cvs file and get normalization constants (CGS units)
    for a gas simulation.
    Input:
        File path (str)
    Output:
        Normalization constants for (CGS units):
            rho: density
            v: velocity
            L: length
            p: thermal pressure
            B: magnetic field
            t: time
    Author: Alan Palma
    """
    #Read with pandas
    data = pd.read_csv(path, sep = ",")

    #Put into pandas objects
    rho = np.array(data.loc[data["variable"]=="rho_0"]["normalisation"])
    v = np.array(data.loc[data["variable"]=="v_0"]["normalisation"])
    L = np.array(data.loc[data["variable"]=="L_0"]["normalisation"])

    #Derive other normalization constants
    p = rho*v**2 #thermal pressure
    B = np.sqrt(4*np.pi*rho*v**2) #magnetic field
    t = L/v #time

    return rho, v, L, p, B, t
```

(b) Create a python function that reads the **vtk.out** file, reads the second column, and returns the times in CGS units using t_0 from point (a).

In [3]:
```python
def io_time_cgs(path, t_0):
    """
    Fuction to get the time simulation in CGS untis.
    Input:
        Path: file path (str)
        t_0: time normalization constant (float)
    Output:
        t_cgs: time array with time simualtion (1D array, float)
    Author: Alan Palma
    """
    data = pd.read_csv(path, sep = "\s+", header = None)

    # Get the second column
    time_code = np.array(data.iloc[:,1], dtype = float)

    # Convert this to CGS units
    t_cgs = time_code*t_0

    return t_cgs
```

(c) Create a python function that reads a VTK data file, normalises the data fields to CGS units using the values from points (a) and (b), and returns them jointly with the mesh and time information as tuple objects.

In [4]:
```python
def io_vtk_file(path, norm_unit, time_arr):
    """
    Function to read a vtk file and extract the data of simulation (in CGS untis).
    Input:
        Path: file directory (str)
        norm_unit: array with all normalization factors (rho_0, v_0, L_0, p_0, B_0,
        time_arr: 1D time array for all simulation data (float)
    Output:
        mesh: pvista object with the vtk file
        rho_cgs_2D: 2D gas density array in CGS units (float)
        vx1_cgs_2D: 2D x velocity array in CGS units (float)
        vx2_cgs_2D: 2D y velocity array in CGS units (float)
        Bx1_cgs_2D: 2D x magnetic field array in CGS units (float)
        Bx2_cgs_2D: 2D y magnetic field array in CGS units (float)
        prs_cgs_2D: 2D gas thermal pressure array in CGS units (float)
        time: information time correponding to simulation in CGS untis (float)
    """
    mesh = pv.read(path)

    #Arrays in code units

    rho = pv.get_array(mesh, "rho", preference = 'cell')
    vx1 = pv.get_array(mesh, "vx1", preference = 'cell')
    vx2 = pv.get_array(mesh, "vx2", preference = 'cell')
    Bx1 = pv.get_array(mesh, "Bx1", preference = 'cell')
    Bx2 = pv.get_array(mesh, "Bx2", preference = 'cell')
    prs = pv.get_array(mesh, "prs", preference = 'cell')

    #Arrays in CGS units

    rho_cgs = rho*norm_unit[0]
    vx1_cgs = vx1*norm_unit[1]
    vx2_cgs = vx2*norm_unit[1]
    Bx1_cgs = Bx1*norm_unit[2]
    Bx2_cgs = Bx2*norm_unit[2]
    prs_cgs = prs*norm_unit[3]


    # 2D arrays in CGS units

    rho_cgs_2D = rho_cgs.reshape(mesh.dimensions[0] - 1, mesh.dimensions[1] - 1)
```

```
    vx1_cgs_2D = vx1_cgs.reshape(mesh.dimensions[0] - 1, mesh.dimensions[1] - 1)
    vx2_cgs_2D = vx2_cgs.reshape(mesh.dimensions[0] - 1, mesh.dimensions[1] - 1)
    Bx1_cgs_2D = Bx1_cgs.reshape(mesh.dimensions[0] - 1, mesh.dimensions[1] - 1)
    Bx2_cgs_2D = Bx2_cgs.reshape(mesh.dimensions[0] - 1, mesh.dimensions[1] - 1)
    prs_cgs_2D = prs_cgs.reshape(mesh.dimensions[0] - 1, mesh.dimensions[1] - 1)

    #Get the time information

    file_name = path[-13:]
    indx = int(file_name[7:9]) #Index from file_name
    time = time_arr[indx]


    return mesh, rho_cgs_2D, vx1_cgs_2D, vx2_cgs_2D, Bx1_cgs_2D, Bx2_cgs_2D, prs_cg
```

(d) Call all the above functions for VTK file # 30 of each simulation, and make the following maps using the correct mesh coordinates, dimensions and time, all in in CGS units:

- A figure showing the gas density, ρ.
- A figure showing the gas sound speed, $c_s=\sqrt{\gamma \frac{p}{\rho}}$.
- A figure showing showing the kinetic energy density, $E_k = \frac{1}{2}\rho\,v^2$ with $v=\sqrt{v_x^2+v_y^2}$.
- A figure showing the magnetic vector field, $\vec{B}= \vec{B}_x + \vec{B}_y$.

Notes: Choose different perceptually-uniform colour schemes for each of the above quantities, and fix the colour bar limits. Add the correct time stamp in CGS units to each map. Since these are high-resolution models, one way to improve the visualisation of 2D vector fields is to interpolate them into a coarser grid.

In [5]:
```
#Call the functions

directory = "Orszag_Tang-MHD/"
file_name = "data.0030.vtk"

#Normalization values

rho_0, v_0, L_0, p_0, B_0, t_0 = io_norm_values(directory + "units.out")

#Time information

t_cgs = io_time_cgs(directory + "vtk.out", t_0)

#Create an array for the normalization constants

norm_arr = np.array([rho_0, v_0, B_0, p_0, L_0, t_0])

#Call the function for .vtk file

mesh, rho_cgs_2D, vx1_cgs_2D, vx2_cgs_2D, \
    Bx1_cgs_2D, Bx2_cgs_2D, prs_cgs_2D, t_cgs_30 = io_vtk_file(directory + file_nam
```

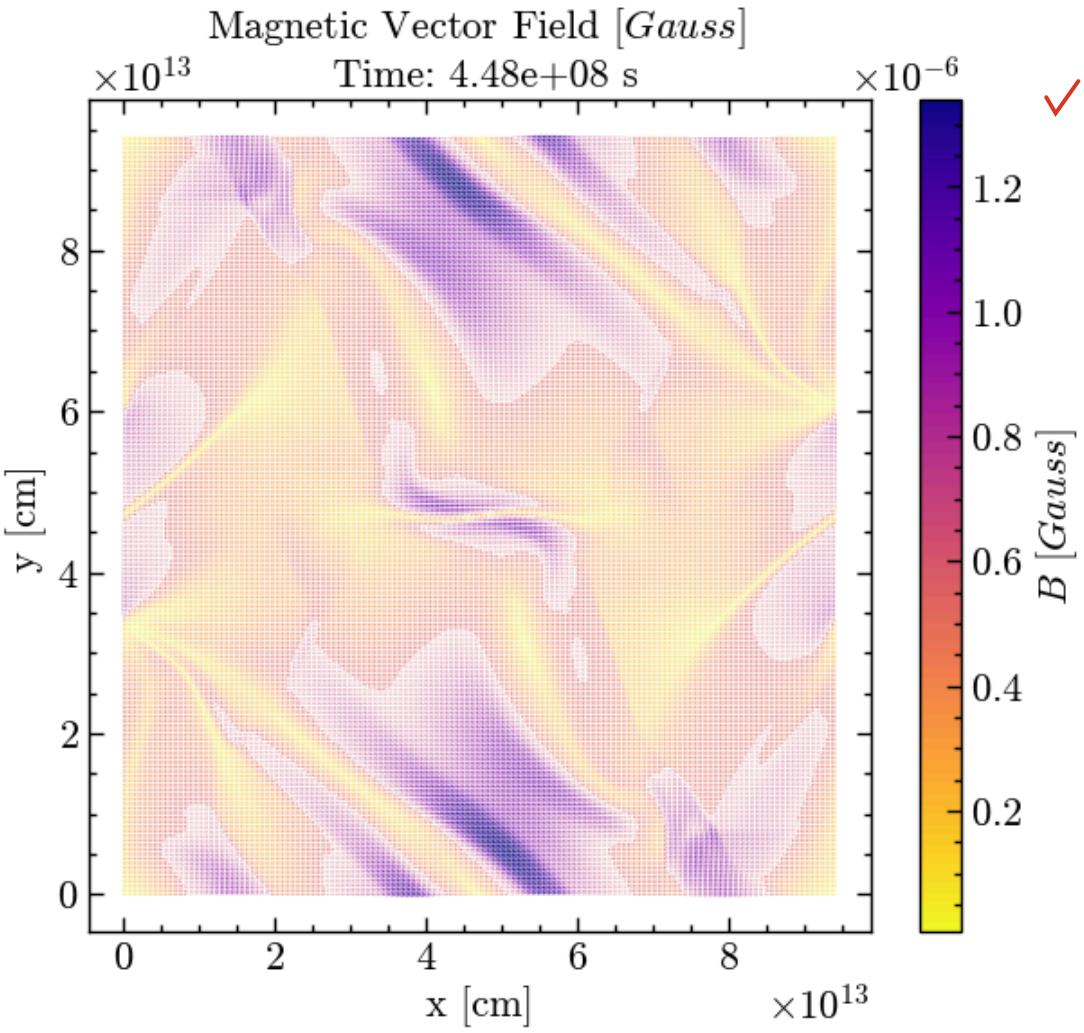
In [6]:
```
# Create coordinate vectors:
x = np.linspace(mesh.bounds[0], mesh.bounds[1], mesh.dimensions[1] - 1)*L_0
y = np.linspace(mesh.bounds[2], mesh.bounds[3], mesh.dimensions[0] - 1)*L_0

# Generate Grid
x_2d, y_2d = np.meshgrid(x, y)

#print(x.shape, y.shape)
```

In [7]:
```
#Compute the gas sound speed

#Adiabatic constant
gamma = 5./3.

cs_cgs_2D = np.sqrt(gamma*(prs_cgs_2D/rho_cgs_2D))

#print(c_s.shape)
```

In [8]:
```
#Velocity magnitude
v_cgs_2D = np.sqrt(vx1_cgs_2D**2+vx2_cgs_2D**2)

#Compute the kinetic energy density
energyK_cgs_2d = 0.5*rho_cgs_2D*v_cgs_2D**2

#print(eneryK_cgs_2d.shape)
```

In [9]:
```
#Magnetic field magnitude

B_cgs_2D = np.sqrt(Bx1_cgs_2D**2+Bx2_cgs_2D**2)

#print(B_cgs_2D.shape)
```

In [10]:
```
#Let's see the vector magnetic flied

plt.style.use(['science', 'notebook', 'no-latex'])

fig = plt.figure(figsize=(7, 6))

plt.quiver(x_2d, y_2d, Bx1_cgs_2D, Bx2_cgs_2D, B_cgs_2D, cmap = "plasma_r")

plt.colorbar(label = "$B$ [$Gauss$]")

plt.title(f"Magnetic Vector Field [$Gauss$] \n Time: {'%.2e' % t_cgs_30} s")
plt.xlabel("x [cm]")
plt.ylabel("y [cm]")

plt.show()
```



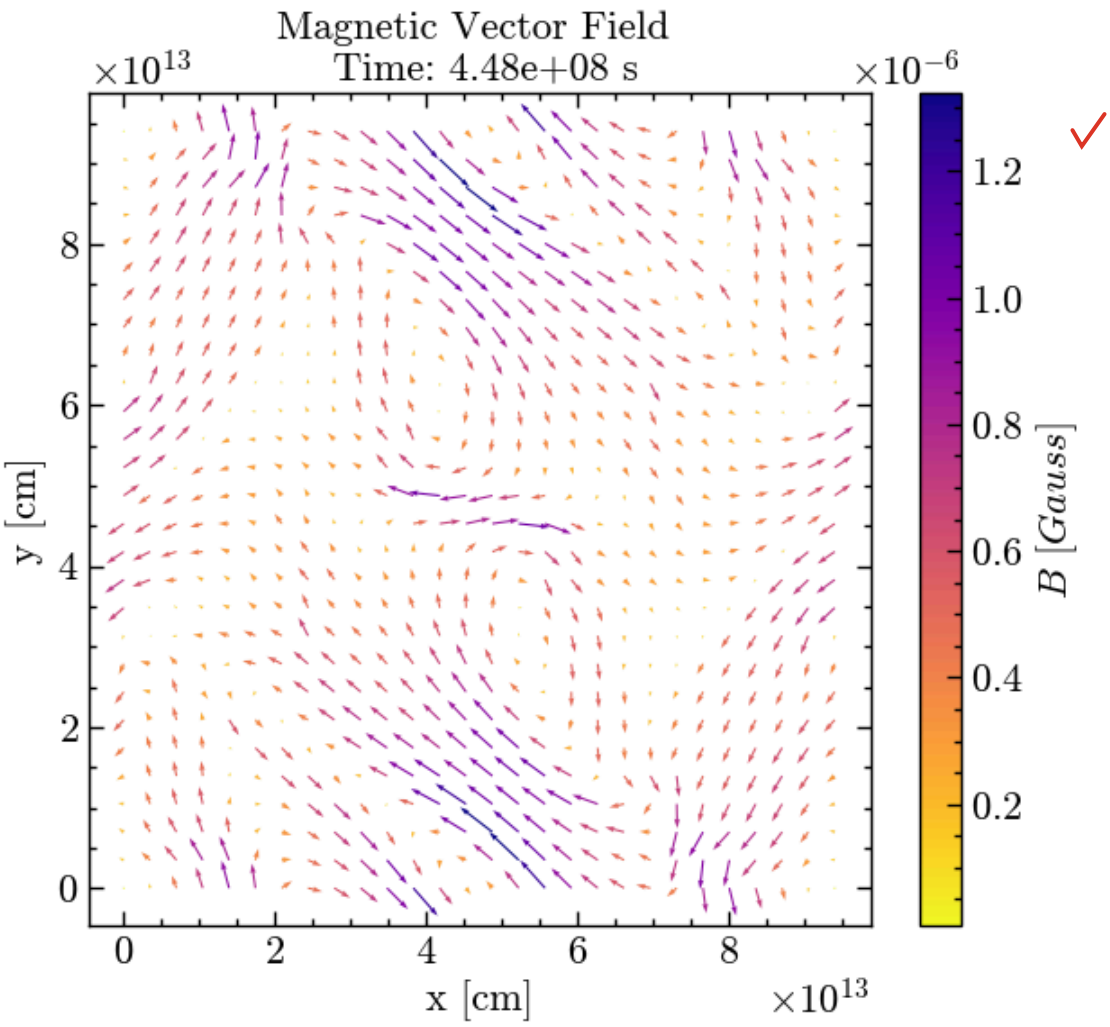
Since there is a lot of data points, it is impossible to see magnetic vector field.

In [11]:
```
from scipy.interpolate import griddata
```

In [12]:
```
#Interpolate the magtetic field

#Set a array with new dimentions
fac_red = 9. #Factor to reduce the number of points
new_dim= np.linspace(mesh.bounds[0], mesh.bounds[1], int((mesh.dimensions[1] - 1)/f

#New grid for interpolated values
new_x_2d, new_y_2d = np.meshgrid(new_dim, new_dim)

#Interpolate
Bx1_intp = griddata((x_2d.flatten() , y_2d.flatten()), Bx1_cgs_2D.flatten(), (new_x
Bx2_intp = griddata((x_2d.flatten() , y_2d.flatten()), Bx2_cgs_2D.flatten(), (new_x

#Interpolated magnetic field modulus
B_intp = np.sqrt(Bx1_intp**2+Bx2_intp**2)

#print(Bx1_intp.shape, B_intp.shape)
```

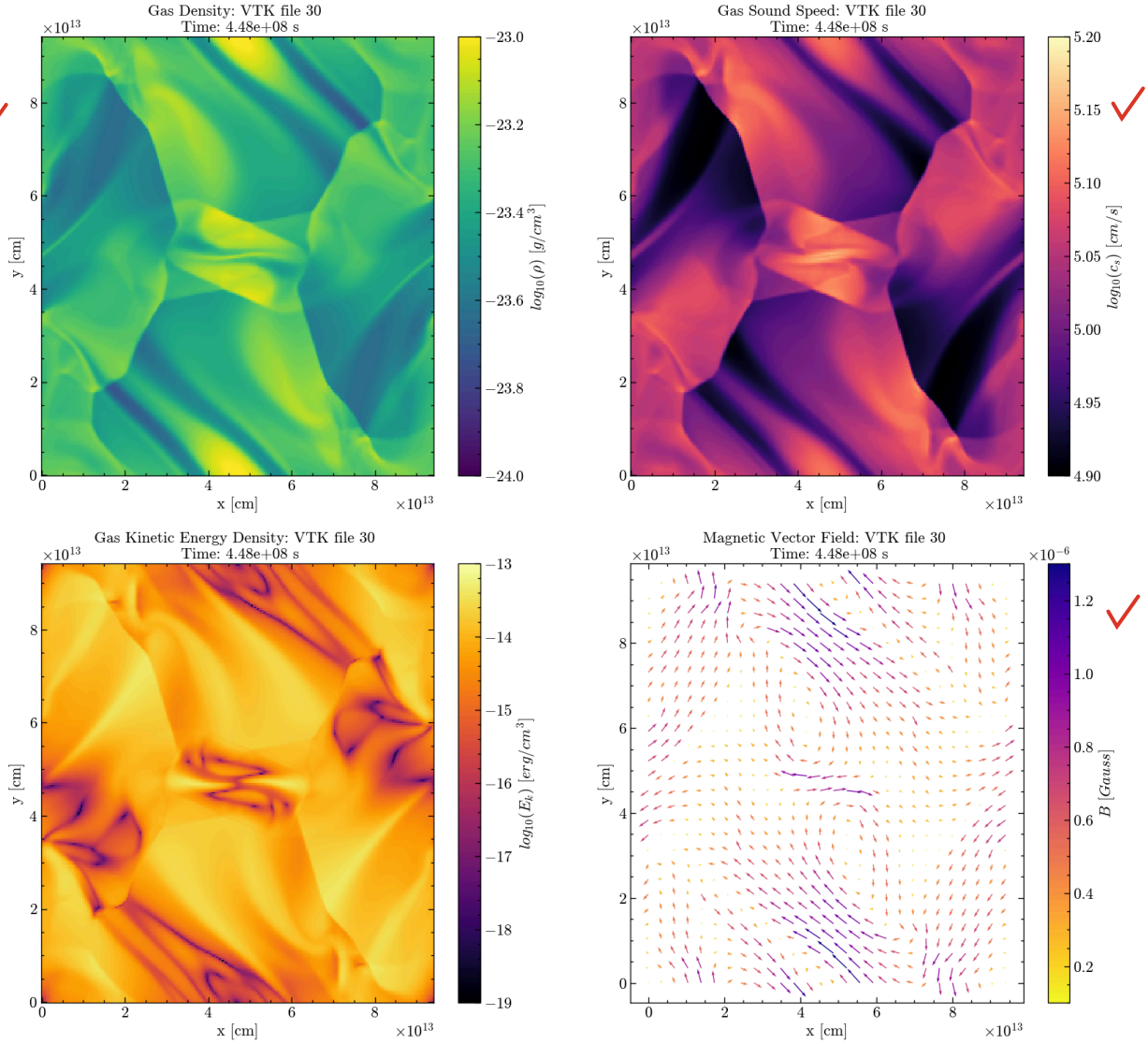
In [13]:
```
#Plotting vector magnetic field interpolated
fig = plt.figure(figsize=(7, 6))

plt.quiver(new_x_2d, new_y_2d, Bx1_intp, Bx2_intp, B_intp, cmap = "plasma_r")

plt.colorbar(label = "$B$ [$Gauss$]")

plt.title(f"Magnetic Vector Field \n Time: {'%.2e' % t_cgs_30} s")
plt.xlabel("x [cm]")
```

```
plt.ylabel("y [cm]")

plt.show()
```



In [14]:
```
#Ploting all together

#Figure enviroment

#Gas density rho
fig, ax = plt.subplots(nrows = 2, ncols = 2, figsize=(20, 18))
ax1, ax2, ax3, ax4 = ax.flatten()

map1 = ax1.pcolormesh(x_2d, y_2d, np.log10(rho_cgs_2D), vmin = -24, vmax=-23.0, cma
ax1.set_title(f"Gas Density: VTK file 30 \n Time: {'%.2e' % t_cgs_30} s")
ax1.set_xlabel("x [cm]")
ax1.set_ylabel("y [cm]")

plt.colorbar(map1, label = "$log_{10}(\\rho)$ [$g/cm^{3}$]")

#Gas Sound velocity
map2 = ax2.pcolormesh(x_2d, y_2d, np.log10(cs_cgs_2D), vmin = 4.9, vmax= 5.2, cmap
ax2.set_title(f"Gas Sound Speed: VTK file 30  \n Time: {'%.2e' % t_cgs_30} s")
ax2.set_xlabel("x [cm]")
ax2.set_ylabel("y [cm]")

plt.colorbar(map2, label = "$log_{10}(c_s)$ [$cm/s$]")

#Gas kinetic energy density
map3 = ax3.pcolormesh(x_2d, y_2d, np.log10(energyK_cgs_2d), vmin = -19., vmax= -13.
ax3.set_title(f"Gas Kinetic Energy Density: VTK file 30 \n Time: {'%.2e' % t_cgs_30
ax3.set_xlabel("x [cm]")
ax3.set_ylabel("y [cm]")
```

```
plt.colorbar(map3, label = "$log_{10}(E_k)$ [$erg/cm^{3}$]")

#Magnetic vector field
# map4 = ax4.pcolormesh(x_2d, y_2d, B_cgs_2D, vmin = 0.1e-6, vmax = 1.3e-6 , cmap =
map5 = ax4.quiver(new_x_2d, new_y_2d, Bx1_intp, Bx2_intp, B_intp, clim = (0.1e-6, 1
ax4.set_title(f"Magnetic Vector Field: VTK file 30 \n Time: {'%.2e' % t_cgs_30} s")
ax4.set_xlabel("x [cm]")
ax4.set_ylabel("y [cm]")

plt.colorbar(map5, label = "$B$ [$Gauss$]")

plt.show()
```



(e) Create a set of Python functions that loops over all the VTK simulation files and returns maps of the density field ρ, the kinetic energy density E_k, and also histograms of the density field in CGS units for all times, into a folder called "output_data".

In [15]:
```
#Third party libraries
import os
```

In [16]:
```
# Create a directory
if os.path.isdir("output_data"):
    print("Directory already exists.")
else:
    print("Directory has been created.")
    os.mkdir("output_data")
```

```
Directory has been created.
```

In [17]:
```python
def plot_map_rho(j, x, y, rho, t):
    """
    Function to plot and save gas density map in CGS units.
    Inputs:
        j: index for looping (int)
        x: 2D  mesh grid in x (float)
        y: 2D mesh grid in y (float)
        rho: 2D array of gas density (float)
        t: information time correponding to simulation (float)
    Output:
        Gas density map figure
    Author: Alan Palma
    """

    plt.figure(figsize=(7,6))

    map1 = plt.pcolormesh(x, y, np.log10(rho), vmin = -24, vmax=-23.0, cmap = "viri
    plt.title(f"Gas Density: VTK File {j} \n Time: {'%.2e' % t} s")
    plt.xlabel("x [cm]")
    plt.ylabel("y [cm]")

    plt.colorbar(map1, label = "$log_{10}(\\rho)$ [$g/cm^{3}$]")

    plt.savefig("output_data/gas_density_simulation/gas_density.{:03d}.png".format(
    plt.close()
```

In [18]:
```python
def plot_map_kineticE(j, x, y, energy, t):
    """
    Function to plot and save gas kinetic energy map.
    Inputs:
        j: index for looping (int)
        x: 2D  mesh grid in x (float)
        y: 2D mesh grid in y (float)
        energy: 2D array of kinetic energy density (float)
        t: information time correponding to simulation (float)
    Output:
        Gas kinetic energy map figure
    Author: Alan Palma
    """

    plt.figure(figsize=(7,6))

    map2 = plt.pcolormesh(x, y, np.log10(energy), vmin = -19., vmax= -13., cmap = "
    plt.title(f"Gas Kinetic Energy Density: VTK File {j} \n Time: {'%.2e' % t} s")
    plt.xlabel("x [cm]")
    plt.ylabel("y [cm]")

    plt.colorbar(map2, label = "$log_{10}(E_k)$ [$erg/cm^{3}$]")

    plt.savefig("output_data/kinetic_energy_simulation/kinetic_energy.{:03d}.png".f
    plt.close()
```

In [19]:
```python
def plot_rho_hist(j, rho, t):
    """
    Function to plot and save gas density histogram.
    Inputs:
        j: index for looping (int)
        rho: 2D array of gas density (float)
        t: information time correponding to simulation (float)
    Output:
        Gas density histogram figure
    Author: Alan Palma
    """
```

```
    plt.figure(figsize=(7,6))

    plt.hist(rho.flatten(), histtype = "step", bins = 40, color = "orange")


    plt.xlim(0.1e-23, 1.2e-23)
    plt.ylim(0., 8.e3)

    plt.grid(True, alpha = 0.3)
    plt.title(f"Gas Density Histogram: VTK File {j} \n Time: {'%.2e' % t} s")
    plt.xlabel("$\\rho$ [$g/cm^{3}$]")
    plt.ylabel("N")

    plt.savefig("output_data/gas_density_hist_simulation/gas_density_hist.{:03d}.pn
    plt.close()
```

In [20]:

```
def graph_simulation(directory, norm_arr, time_arr):
    """
    Function that opens all the simulation files .vtk from a directory,
    and plot gas density, kinetic energy density, and gas density histogram.
    Inputs:
        directory: path directory where the .vtk file are stored
        norm_arr: array with all normalization factors (rho_0, v_0, L_0, p_0, B_0,
        time_arr: 1D time array for all simulation data (float)
    Outputs:
        Figures corresponding to gas density, kinetic energy density, and gas densi
    Author: Alan Palma
    """

     #Create a folder to store gas density figures

    if os.path.isdir("output_data/gas_density_simulation"):
        print("Directory already exists.")
    else:
        print("Directory has been created.")
        os.mkdir("output_data/gas_density_simulation")

    #Create a folder to store gas kinetic energy density figures

    if os.path.isdir("output_data/kinetic_energy_simulation"):
        print("Directory already exists.")
    else:
        print("Directory has been created.")
        os.mkdir("output_data/kinetic_energy_simulation")

    #Create a folder to store gas density histograms

    if os.path.isdir("output_data/gas_density_hist_simulation"):
        print("Directory already exists.")
    else:
        print("Directory has been created.")
        os.mkdir("output_data/gas_density_hist_simulation")

    #For loop to go over all simulation files
    for j in range(0, len(time_arr)):

        filename = directory + "data.0{:03d}.vtk".format(j)

        #Call the fuction to get all data
        mesh, rho_cgs_2D, vx1_cgs_2D, vx2_cgs_2D, \
        _ , _ , _ , t_cgs_sim = io_vtk_file(filename, norm_arr, time_arr)

        # Create coordinate vectors:
        x = np.linspace(mesh.bounds[0], mesh.bounds[1], mesh.dimensions[1] - 1)*L_0
        y = np.linspace(mesh.bounds[2], mesh.bounds[3], mesh.dimensions[0] - 1)*L_0
```

```
        # Generate Grid
        x_2d, y_2d = np.meshgrid(x, y)

        #Speed magnitude
        v_cgs_2D = np.sqrt(vx1_cgs_2D**2+vx2_cgs_2D**2)

        #Compute the kinetic energy density
        energyK_cgs_2d = 0.5*rho_cgs_2D*v_cgs_2D**2

        #Plot and store gas density for all times
        plot_map_rho(j, x_2d, y_2d, rho_cgs_2D, t_cgs_sim)

        #Plot and store gas kinetic energy density for all times
        plot_map_kineticE(j, x_2d, y_2d, energyK_cgs_2d, t_cgs_sim)

        #Plot and store gas density histogram
        plot_rho_hist(j, rho_cgs_2D, t_cgs_sim)

    #Return the last gas density data

    return rho_cgs_2D
```

In [21]:
```
#Call the function

rho_cgs_2D60 = graph_simulation(directory, norm_arr, t_cgs)
```

```
Directory has been created.
Directory has been created.
Directory has been created.
```

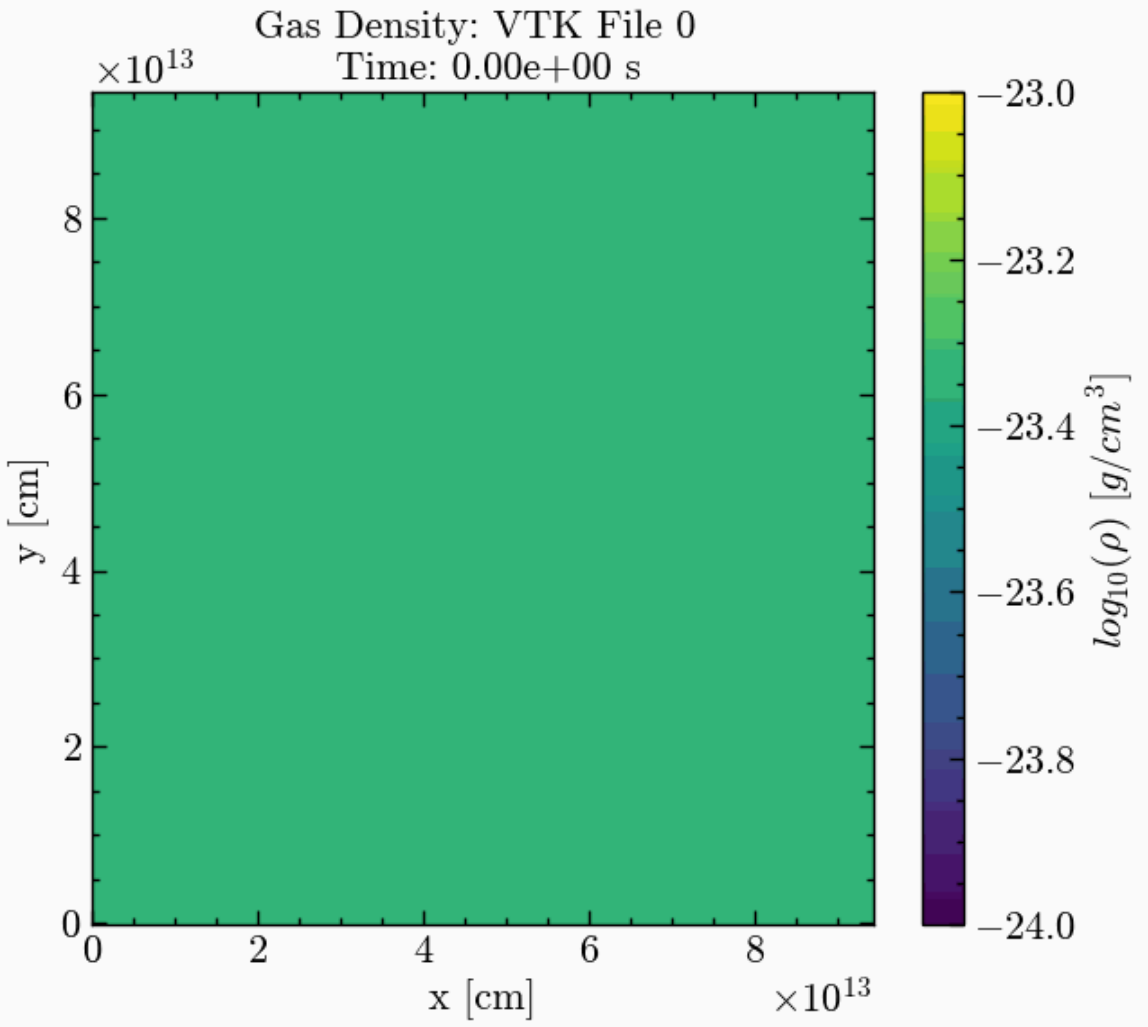
(f) Briefly describe: what happens with the density field as time progresses? Does the density field follow any statistical distribution at late times?

In [24]:
```
from PIL import Image
import glob
from IPython import display
```

In [25]:
```
#Create an animation to see density field

images_input = "output_data/gas_density_simulation/gas_density.***.png"
imgif_output = "output_data/gas_density_simulation/gas_density.gif"

# Collect the images
imgs = (Image.open(f) for f in sorted(glob.glob(images_input)))

img = next(imgs)

img.save(fp = imgif_output, format="GIF", append_images=imgs,\
        save_all=True, duration = 100, loop = 0)
```

In [26]:
```
#Show the animation

display.Image(open('output_data/gas_density_simulation/gas_density.gif','rb').read(
```

Out[26]:



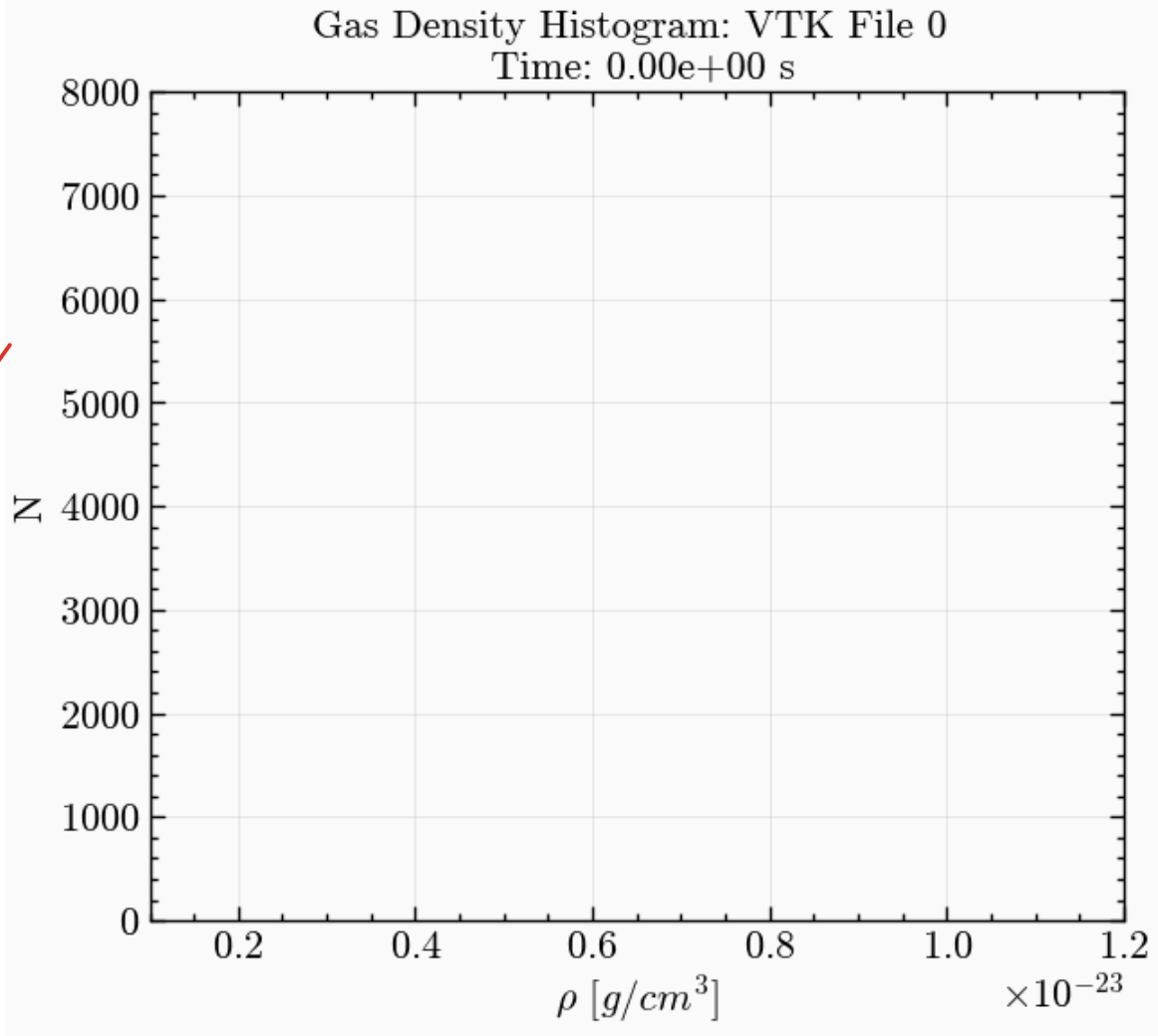
In [27]:

```
#Create an animation to see density field histogram evolution

images_input = "output_data/gas_density_hist_simulation/gas_density_hist.***.png"
imgif_output = "output_data/gas_density_hist_simulation/gas_density_hist.gif"

# Collect the images
imgs = (Image.open(f) for f in sorted(glob.glob(images_input)))

img = next(imgs)

img.save(fp = imgif_output, format="GIF", append_images=imgs,\
        save_all=True, duration = 100, loop = 0)
```

In [28]:

```
#Show the animation

display.Image(open('output_data/gas_density_hist_simulation/gas_density_hist.gif','
```

Out[28]:

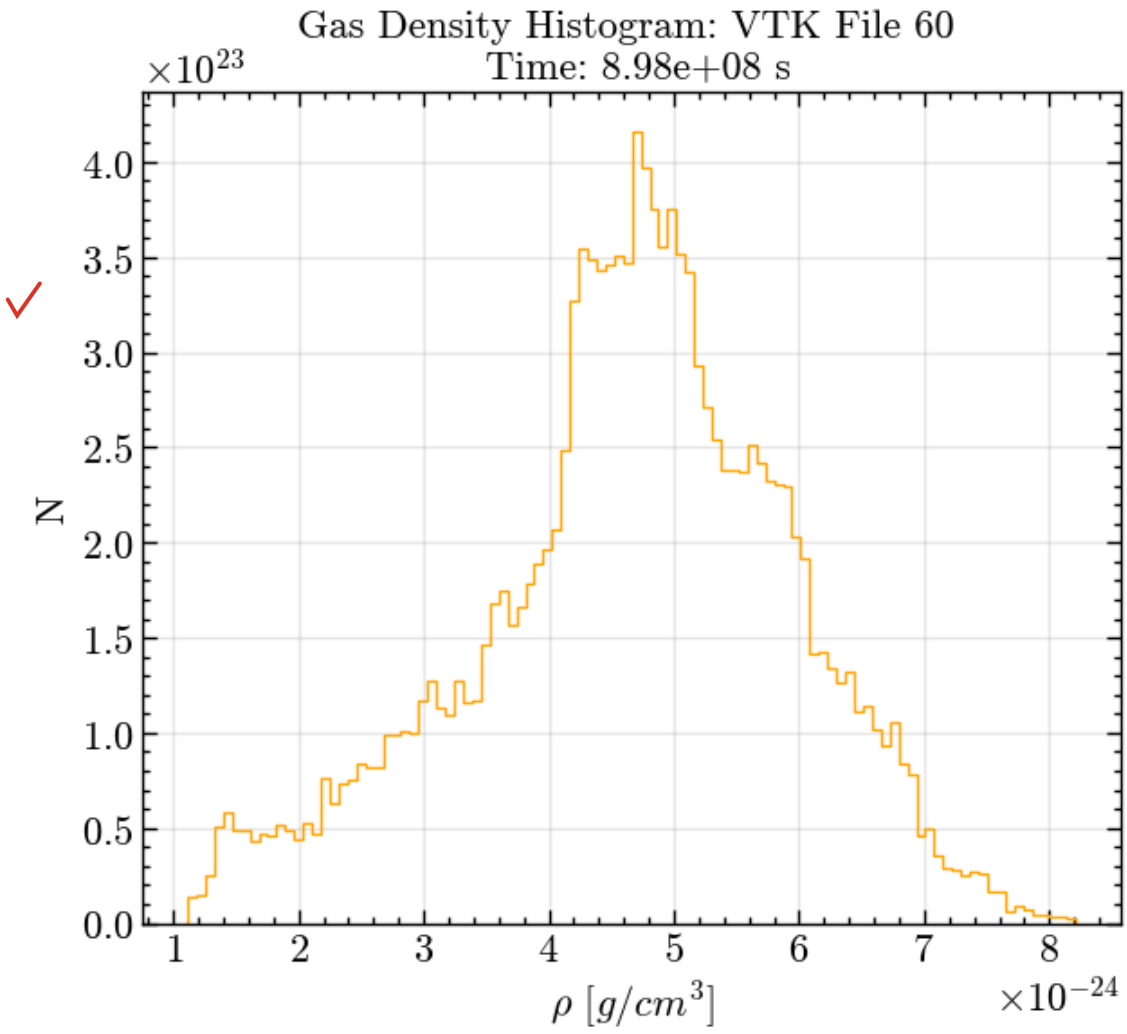


- It is observed that the initially uniform gas energy density changes over time, showing turbulence. This is due to the initial magnetic and velocity conditions. Actually, the swirling pattern in the initial magnetic field conditions is reflected in the rapid development of turbulence in the gas density.
- The gas density distribution appears to follow a Gaussian-like distribution at the final time. ok.

Fit a Gaussian function usign regresion methods

In [29]:
```
# print(rho_cgs_2D60.shape)
```

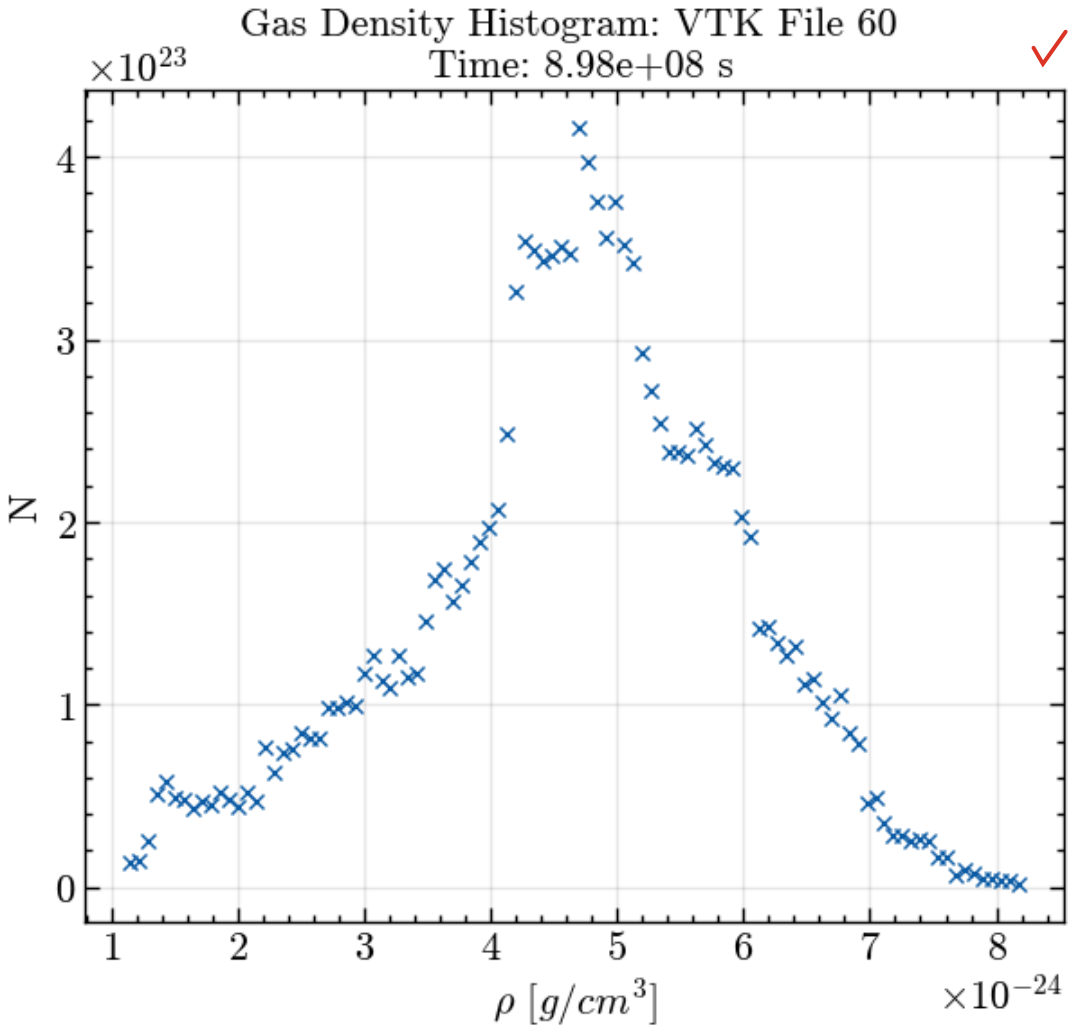
In [30]:
```
#From the last simulation data (vtk60)
#Histogram

plt.figure(figsize=(7,6))

n, bins, _ = plt.hist(rho_cgs_2D60.flatten(), histtype = "step", bins = 100, densit

# plt.xlim(0.1e-23, 1.2e-23)
# plt.ylim(0., 8.e3)

plt.grid(True, alpha = 0.3)
plt.title(f"Gas Density Histogram: VTK File 60 \n Time: {'%.2e' % t_cgs[-1]} s")
plt.xlabel("$\\rho$ [$g/cm^{3}$]")
plt.ylabel("N")

plt.show()
```



In [31]:
```
# print(n.shape)
# print(bins.shape)
```

In [32]:
```
# Shift arrays:
x0 = 0.5 * ( bins[1:] + bins[:-1] )

# print(x0.shape)
```

In [33]:
```
# New histogram plot:

plt.figure(figsize=(7,6))

plt.plot(x0, n, lw=3.0, marker = "x", linestyle= " ")

plt.grid(True, alpha = 0.3)
plt.title(f"Gas Density Histogram: VTK File 60 \n Time: {'%.2e' % t_cgs[-1]} s")
plt.xlabel("$\\rho$ [$g/cm^{3}$]")
plt.ylabel("N")

plt.show()
```



A good model is a normal distro:

$$\rho_x=\frac 1 {\sigma\sqrt{2\pi}}\ e^{- \frac{1}{2}(\frac{x - \mu}{\sigma})^2}$$

In [34]:
```
# Define the model for the fit

def log_normal(x, mu, s):
    """
    Function to define a normal distro.
    Inputs:
        x: 1D array with the independent variable (float)
        mu: mean (float)
        s: standard deviation (float)
    Outputs:
        y_model: 1D array with the dependent variable (float)
    Author: Alan Palma
    """

    y_model = (1/(s*np.sqrt(2*np.pi)))*(np.exp(-0.5*((x-mu)/s)**2))

    return y_model
```

In [35]:
```
import scipy.optimize as opt
```

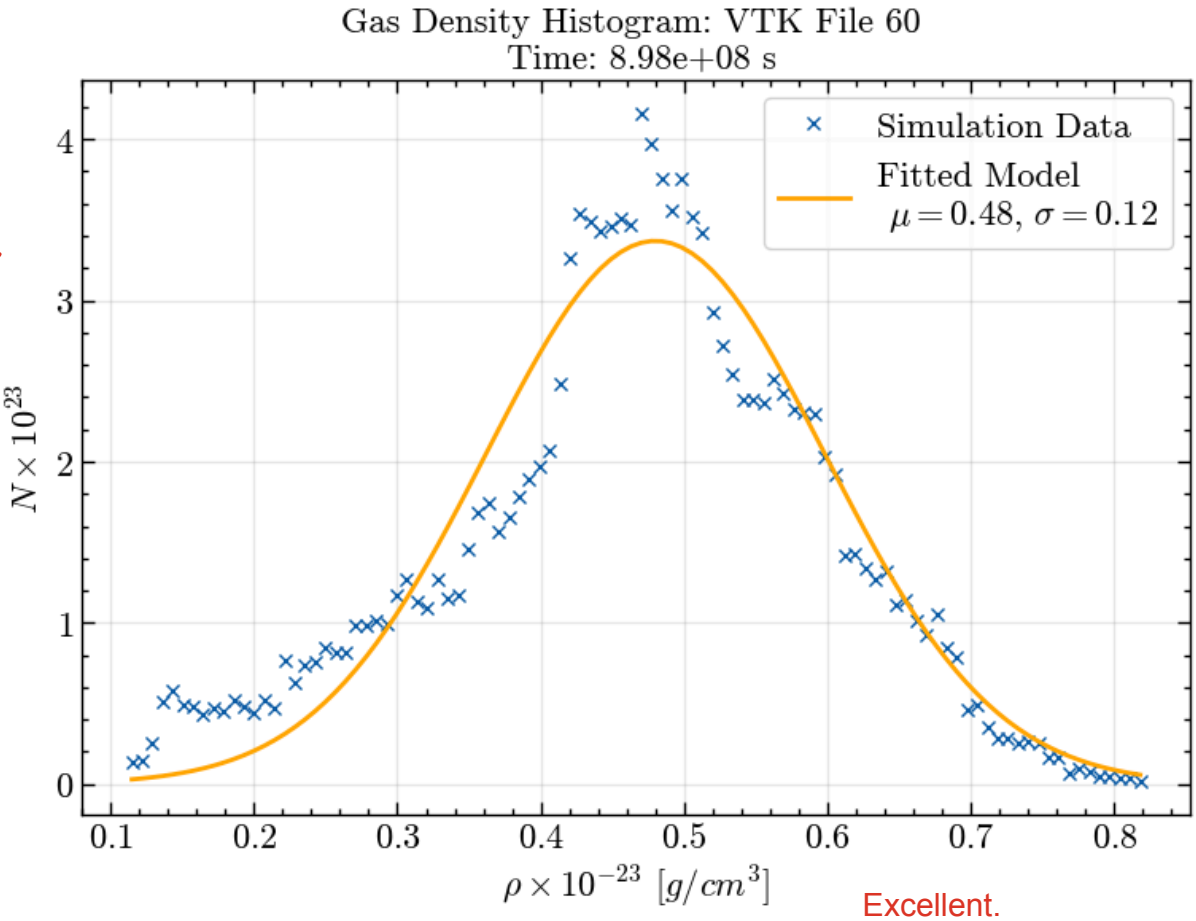
In [36]:
```
# Renormalise the axes

n1 = n/1.e23
x1 = x0/1.e-23
```

In [37]:
```
# Fitting
```

```
coef, cova = opt.curve_fit(log_normal, x1, n1)

#print(coef)
```

In [38]:
```
#Evaluate the distro with the fitting parameters

n_fit = log_normal(x1, coef[0], coef[1])
```

In [39]:
```
# Plot the fitted model

plt.figure(figsize=(9,6))

plt.plot(x1, n1, lw=3.0, marker = "x", linestyle= " ", label = "Simulation Data")
plt.plot(x1, n_fit, color = "orange", label = f"Fitted Model \n $\\mu ={'%.2f' % co

plt.grid(True, alpha = 0.3)
plt.title(f"Gas Density Histogram: VTK File 60 \n Time: {'%.2e' % t_cgs[-1]} s")
plt.xlabel("$\\rho \\times 10^{-23}$ [$g/cm^{3}$]")
plt.ylabel("$N\\times 10^{23}$")
plt.legend(frameon = True, loc = 1)

plt.show()
```



Excellent.

(g) Create a set of Python functions that loops over all the VTK simulation files, computes the following quantities in CGS units for each time:

- the average gas temperature, $\overline T$, (**Hint:** the temperature in each grid cell can be calculated using the equation of state for ideal gases, i.e., $p=\frac{\rho k_B\,T}{\mu m_u}$, where k_B is the Boltzmann constant, m_u is the atomic mass unit, and $\mu=0.6$ is the mean particle mass in the gas.)
- the average kinetic energy density, $\overline E_k$, where $E_k = \frac{1}{2}\rho\,v^2$.

- the average magnetic energy density, $\overline E_m$, where $E_m = \frac{1}{2}\frac{B^2}{\,\mu_0}$, where $\mu_0\equiv$ magnetic permeability of free space.

and returns:

- a CSV file with 4 columns, time on the first column, and the above quantities in the next ones. The CSV file should be named "stats.csv" saved into the folder called "output_data".

Dimetional analisis:

- Average Temperature:

  $$\bar{T} = \frac{\mu m_{\mu} p}{\rho k_{B}}$$

  Since simulation ensure the correct untis for ρ, and p, it is needed to check only for m_{μ}, and k_{B}:

  $$ [k_{B}] = \frac{J}{K} = \frac{kg \; m^2 \; s^{-2}}{K} = \frac{kg \; m^{2} \; s^{-2}}{K} * \frac{10^{4}cm²}{1 m^2} * \frac{10³ g}{1 kg} = 10⁷ \frac{erg}{K}\\ [m_{\mu}] = kg = kg*\frac{10³g}{1kg} = 10^3 g $$

- Magnetic energy density:

  Since it is needed the density energy in CGS units, it is used (with $\mu_0 = 1.0$): $$ \bar{E}_m = \frac{1}{8 \pi} B² $$

  Then,

  $$ [\bar{E}_m] = [\bar{E}_k] = \frac{erg}{cm³} $$

In [40]:
```python
def mean_temp(p, rho):
    """
    Function to calculate mean temperature.
    Inputs:
        p: 2D array of thermal presure in CGS untis (float)
        rho: 2D array of gas density in CGS untis (float)
    Output:
        Temp_mean: average temperature in CGS untis (float)
    Author: Alan Palma
    """
    #Define constants
    mu = 0.6 #mean particle mass in [g]
    m_u = const.atomic_mass * 1e3 #atomic mass unit in [g]
    Kb = const.k * 1e7 #Boltzman constant in [erg/k]

    #Calculate temperature
    Temp = (p*mu*m_u)/(rho*Kb) #Temperature in [K]

    #Mean temperature
    Temp_mean = np.mean(Temp) #Average Temperature in [K]

    return Temp_mean
```

In [41]:
```python
def mean_kineticE(rho, vx1, vx2):
    """
    Function to calculate the mean kinetic energy
    Inputs:
        rho: 2D array of gas density in CGS untis (float)
        vx1_cgs_2D: 2D x velocity array in CGS units (float)
        vx2_cgs_2D: 2D y velocity array in CGS units (float)
    Output:
        energyK_mean: average kinetic energy density in CGS untis (float)
```

```
    Author: Alan Palma
    """
    #Speed magnitude
    v = np.sqrt(vx1**2+vx2**2)

    #Compute the kinetic energy density
    energyK = 0.5*rho*v**2

    #Mean kinetic energy
    energyK_mean = np.mean(energyK) #Mean kinetic energy density [erg/cm^3]

    return energyK_mean
```

In [42]:
```
def mean_magneticE(Bx1, Bx2):
    """
    Function to calculate mean magnetic energy density.
    Inputs:
        Bx1_cgs_2D: 2D x magnetic field array in CGS units (float)
        Bx2_cgs_2D: 2D y magnetic field array in CGS units (float)
    Output:
        magneticE_mean: average magnetic energy density in CGS untis (float)
    """
    #Define constant
    #mu_0 = 1. #Magnetic permeability of free space in CGS

    #Magnetic field magnitude
    B = np.sqrt(Bx1**2+Bx2**2)

    #Magnetic energy density
    magneticE =  B**2 / (8. * np.pi)

    #Mean Magnetic energy density
    magneticE_mean = np.mean(magneticE) #Mean magnetic energy density [erg/cm^3]

    return magneticE_mean
```

In [43]:
```
def stats_csv(directory, time_arr):
    """
    Function to create a .csv file with the stats of the simulation.
    Inputs:
        directory: path directory where the .vtk file are stored (str)
        time_arr: 1D time array for all simulation data (float)
    Output:
        save the .csv file in the direcory "output_data" (pandas object)
    Author: Alan Palma
    """
    #Empty arrays
    temp_stats = []
    kineticE_stats = []
    magneticE_stats = []

    for j in range(0, len(time_arr)):

        filename = directory + "data.0{:03d}.vtk".format(j)

        #Call the fuction to get all data
        mesh, rho_cgs_2D, vx1_cgs_2D, vx2_cgs_2D, \
            Bx1_cgs_2D, Bx2_cgs_2D, prs_cgs_2D, t_cgs_sim = io_vtk_file(filename, n


        #Call function for mean temperature
        T_mean = mean_temp(prs_cgs_2D, rho_cgs_2D)
        temp_stats.append(T_mean)

        #Call function for mean kinetic energy density
        kineticE_mean = mean_kineticE(rho_cgs_2D, vx1_cgs_2D, vx2_cgs_2D)
```

```
        kineticE_stats.append(kineticE_mean)

        #Call function for mean magnetic energy density
        magneticE_mean = mean_magneticE(Bx1_cgs_2D, Bx2_cgs_2D)
        magneticE_stats.append(magneticE_mean)

    #Create a new pandas data frame

    data_frame = pd.DataFrame({"Time [s]" : np.array(time_arr),
                               "Mean Temperature [K]" : np.array(temp_stats),
                               "Mean Kinetic E. density [erg/cm^3]" : np.array(kine
                               "Mean Magnetic E. density [erg/cm^3]" : np.array(mag

    #Export data_frame to csv file
    data_frame.to_csv("output_data/stats.csv", sep = "," , float_format = "{:.6e}".
```

In [44]:
```
#Call the function

directory = "Orszag_Tang-MHD/"

stats_csv(directory, t_cgs)
```

(h) Create a Python function that reads in the CSV file created in (g) and returns (i.e. shows or saves) high-quality labeled figures of each of the above-computed quantities versus time, into the folder called "output_data".

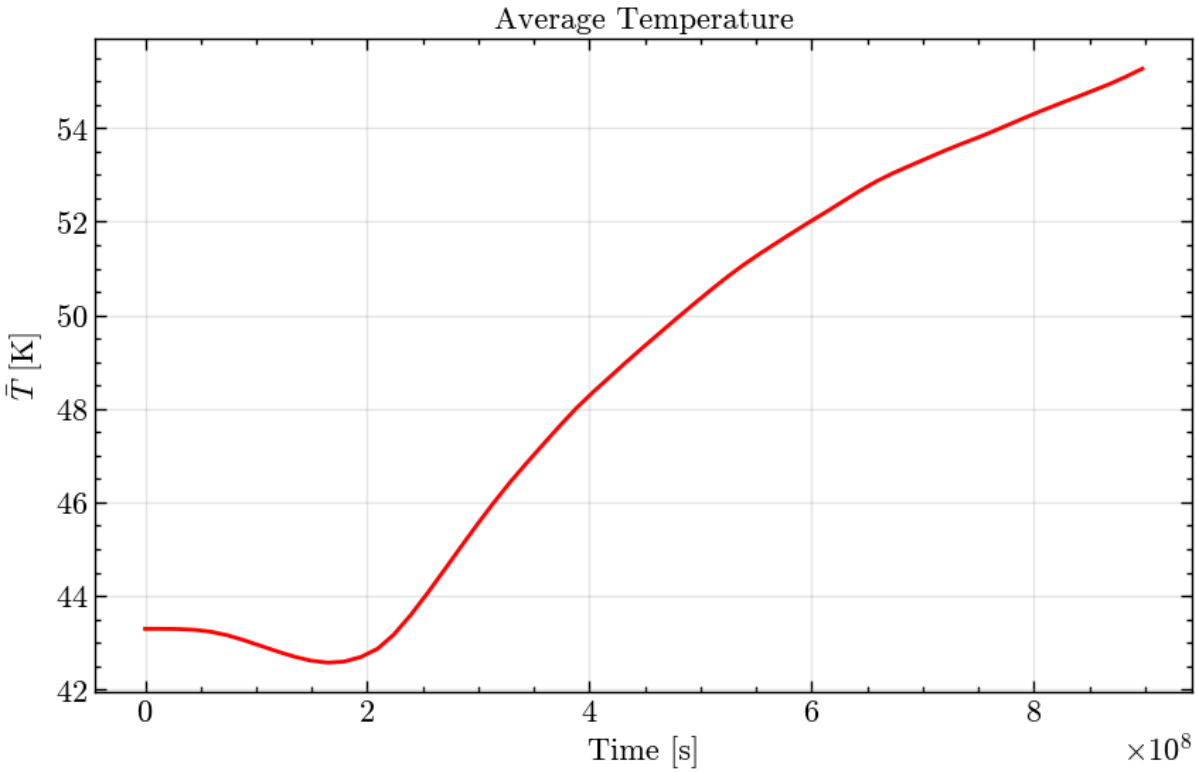
In [45]:
```
# Create a directory to save stats
if os.path.isdir("output_data/stats_simulation"):
    print("Directory already exists.")
else:
    print("Directory has been created.")
    os.mkdir("output_data/stats_simulation")
```

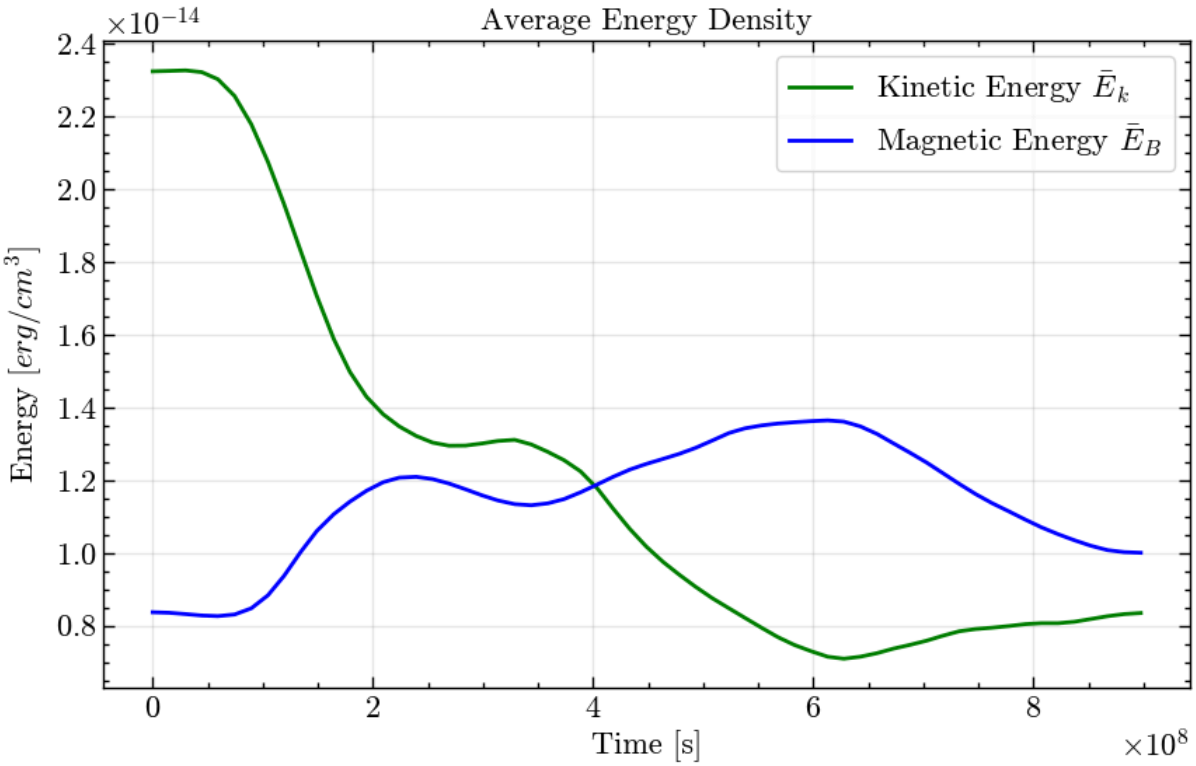
```
Directory has been created.
```

In [46]:
```
def io_csv_figure(directory):
    """
    Fuction to read a csv file and show its data in figures.
    Inputs:
        directory: path directory where the .vtk file are stored (str)
    Outputs:
        data_stats: pandas data frame with stas information
        kineticE_density: 1D array of mean kinetic energy density for all time simu
    Author: Alan Palma
    """

    #Read the data using pandas
    data_stats = pd.read_csv(directory + "stats.csv", sep = ",")

    #Put the information in arrays
    time = np.array(data_stats["Time [s]"])
    Temp =  np.array(data_stats["Mean Temperature [K]"])
    kineticE_density = np.array(data_stats["Mean Kinetic E. density [erg/cm^3]"])
    magneticE_density = np.array(data_stats["Mean Magnetic E. density [erg/cm^3]"])

    #Plot mean temperature
    plt.figure(figsize=(10,6))

    plt.plot(time, Temp, color = "red")

    plt.title("Average Temperature")
    plt.ylabel("$\\bar{T}$ [K]")
    plt.xlabel("Time [s]")
    plt.grid(True, alpha = 0.3)
```

```
    plt.savefig("output_data/stats_simulation/mean_temperature.png")
    plt.show()

    #Plot mean kinetic energy density and mean magnetic energy density


    plt.figure(figsize=(10,6))

    plt.plot(time, kineticE_density, color = "green", label = "Kinetic Energy $\\ba
    plt.plot(time, magneticE_density, color = "blue", label = "Magnetic Energy $\\b

    plt.title("Average Energy Density")
    plt.ylabel("Energy [$erg/cm^{3}$]")
    plt.xlabel("Time [s]")
    plt.legend(frameon = True)
    plt.grid(True, alpha = 0.3)

    plt.savefig("output_data/stats_simulation/mean_energy_density.png")
    plt.show()

    return data_stats, kineticE_density
```

In [47]:
```
#Call the function

directory_data = "output_data/"

data_frame, kineticEnergy_mean = io_csv_figure(directory_data)
```





In [48]:
```
#print(data_frame)
```

(i) Briefly describe: Does the flow reach steady state? Which energy density is dominant?

- Flow does not reach a steady state since temperature and energies do not mantain relatively constant at late times. The mean kinetic energy appears to remain constant at later times, but the magnetic energy density does not. The temperature also shows a continuous increase over time.
- The magnetic energy density is dominant all the time range simulation.

(j) Create a Python function that returns movies showing the time evolution of the kinetic energy density maps computed in (d) and their average values calculated in (g). The movies should be saved into the folder called "output_data".

In [49]:
```
#Create a function to animate

def get_movie(input_imgs, output_movie):
    """
    Fuction that takes images from "imput_imgs" directory and create a movie in "ou
    Input:
        input_imgs: input directory where all images are stored
        output_movie: output directory where the movie (.gif) should be saved
    Output:
        Movie (.gif) created and save in output_movie
    Author: Alan Palma
    """
    images_input = "output_data/" + input_imgs
    imgif_output = "output_data/" + output_movie

    # Collect the images
    imgs = (Image.open(f) for f in sorted(glob.glob(images_input)))

    img = next(imgs)

    img.save(fp = imgif_output, format="GIF", append_images=imgs,\
             save_all=True, duration = 100, loop = 0)
```

In [50]:
```
# Create a directory to save mean kinetic energy plots for movie
if os.path.isdir("output_data/mean_kinetic_energies"):
    print("Directory already exists.")
else:
    print("Directory has been created.")
    os.mkdir("output_data/mean_kinetic_energies")
```

```
Directory has been created.
```

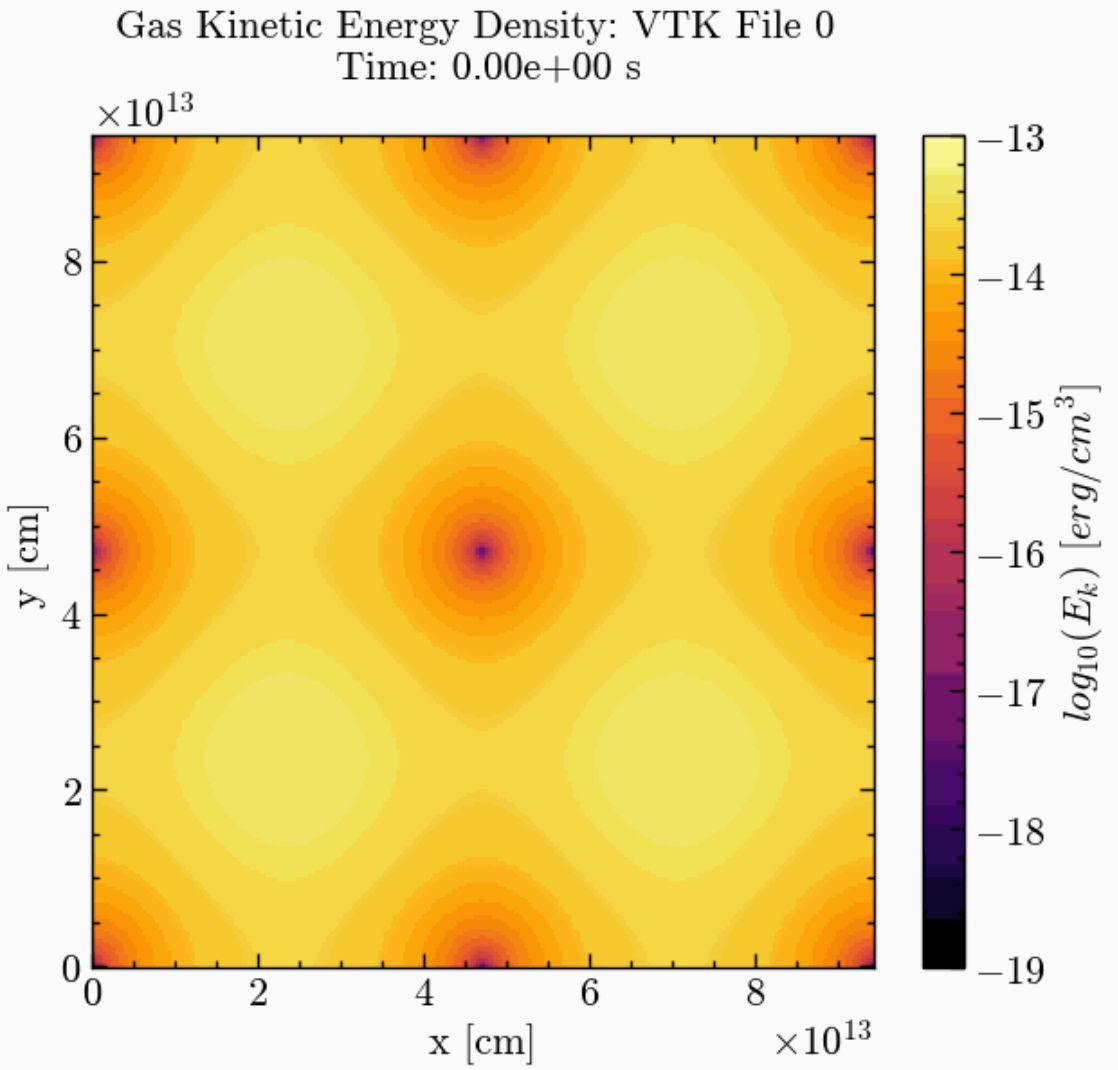
In [51]:
```
def plot_mean_kineticE(j, time, kineticE_density):
    """
    Function to plot and save figures for mean kinetic energy density vs. time.
    Inputs:
        j: index for looping (int)
        time: 1D time array for all simulation data (float)
        kineticE_density: 1D array of average kinetic energy density (float)
    Output:
        Figure of average kinetic energy in fuction of time.
    """
    plt.figure(figsize=(15,6))

    plt.plot(time, kineticE_density , marker = "D", color = "green", mfc = "lawngre
            , label= "$\\bar{E}_K$")

    plt.title(f"Mean Kinetic Energy Density: VTK File {j} \n Time: {'%.2e' % time[-
    plt.ylabel("$\\bar{E}_K$ [$erg/cm^{3}$]")
    plt.xlabel("Time [s]")

    plt.xlim(0., 9.2e8 )
    plt.ylim(0, 2.4e-14)

    plt.legend(frameon = True, loc = 3)
    plt.grid(True, alpha = 0.3)

    plt.savefig("output_data/mean_kinetic_energies/mean_kineticE_density.{:03d}.png
    plt.close()
```

In [52]:
```
#For loop to generate all images needed for mean kinetic energy density movie

for i in range(0, len(t_cgs)):

    #Get arrays from beginning to i+1 element
    time_arr = t_cgs[:i+1]
    KE_mean = kineticEnergy_mean[:i+1]

    #Call the function to plot
    plot_mean_kineticE(i, time_arr, KE_mean)
```

In [53]:
```
#Define input directory and output file name

kinetic_directory = "kinetic_energy_simulation/kinetic_energy.***.png"
kinetic_file_movie = "kinetic_energy.gif"

#Call the fuction to generate the movie for kinetic energy density

get_movie(kinetic_directory, kinetic_file_movie)
```

In [54]:
```
#Show the animation

display.Image(open('output_data/kinetic_energy.gif','rb').read())
```

Out[54]:



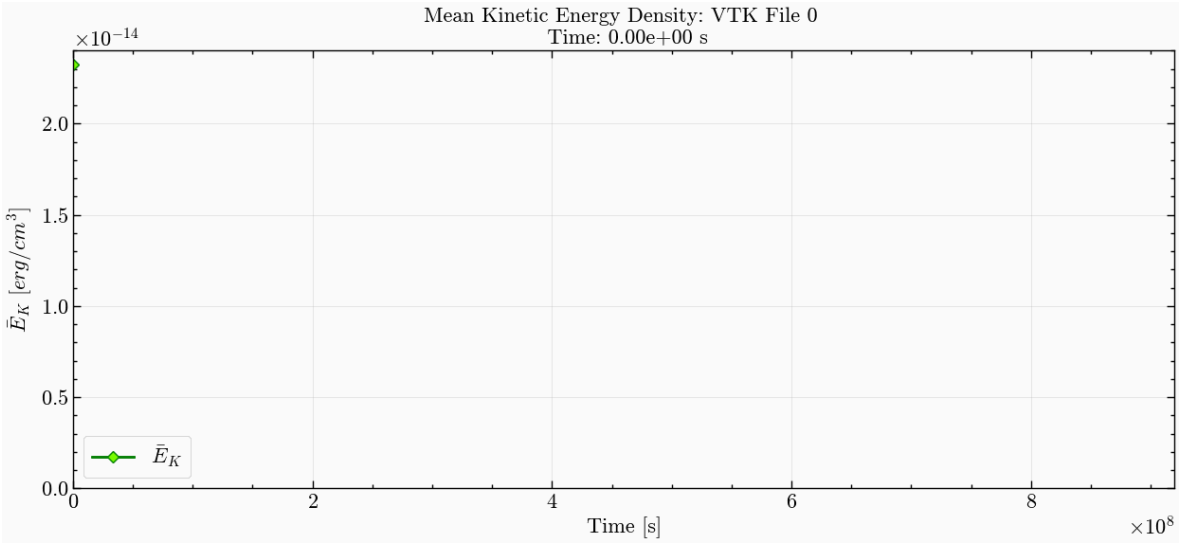
In [55]:

```
#Define input directory and output file name

mean_kinetic_directory = "mean_kinetic_energies/mean_kineticE_density.***.png"
mean_kinetic_file_movie = "mean_kinetic_energy.gif"

#Call the fuction to generate the movie for kinetic energy density

get_movie(mean_kinetic_directory, mean_kinetic_file_movie)
```

In [56]:

```
#Show the animation

display.Image(open('output_data/mean_kinetic_energy.gif','rb').read())
```

Out[56]:


Mean Kinetic Energy Density: VTK File 0
Time: 0.00e+00 s
$\times 10^{-14}$
2.0
1.5
1.0
0.5
0.0
$\bar{E}_K\ [erg/cm^3]$
$\bar{E}_K$
0
2
4
6
8
Time [s]
$\times 10^8$

In [ ]: