Magnetic Mirror Effect in Magnetron Plasma

18BEM0145 - Sashi Kant Shah

18BME2104 - Kaushal Timilsina

18BME2109 - Hrishav Mishra

B.Tech. Mechanical Engineering

Capstone Project Presentation

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Mechanical Engineering

Guide:

Prof. / Dr. Sitaram Dash

Date of Presentation: 03/30/2022

Project Status

Review I

- *Studied Plasma physics*
- *Set up the Problem*
- *Set up the algorithm*

Review II:

- *Running Plasma Simulation*
- *Checks:*
  - *Sampling check*
  - *Update check*
- *Study 1*
  - *Varying Electric Field*
  - *Constant Magnetic Field*
  - *Plots and Animations*

Review III

- *Study 2*
  - *Parabolic Density Function*
  - *Radial Electric Field Configuration (Mimicing an Electrode)*
  - *Helmholtz coil magnetic field configuration /*
  - *Magnetic Mirror configuration(parabolic B along z-axis configuration for $\nabla B \parallel B$)*
  - *Plots and Animations*

Checks

To verify that our program is working correctly, we perform some checks.

1. Sampling
2. Update

1.1 Sampling

- As of now we our sampling of particle speeds is based on the Maxwell-Boltzmann distribution, that we have defined previously. We now check if the average speeds of the particles agrees with the calculations done by hand based on the plasma temperature. We recall the Maxwellian density function that we defined earlier.
- We take the following data and sample 100 particles

Temperature (T) = 10,000 K

Relative Atomic weight of Hydrogen(m) =1.008 g/mol

Universal Gas Constant (R) = 0.831 JK-1mol-1

- For the Hydrogen atom particle distribution, we get

$$\langle v \rangle = \frac{2}{\sqrt{\pi}} \sqrt{\frac{2 \cdot 8.31446261815324 \text{ J K}^{-1} \text{ mol}^{-1} \, 10000 \text{ K}}{1.008 \times 10^{-3} \text{ kg mol}^{-1}}} = 14492.952993825973 \text{ ms}^{-1}$$

Initialization

```
# c1Maxwell means checking the Maxwellian sampling
c1Maxwell = Run()
# Create 100 particles based on the data available in the files
c1Maxwell.create_batch_with_file_initialization('H+', constants.constants['e'][0],
↪  constants.constants['m_H'][0] * constants.constants['amu'][0], 100, 100, 'H ions',
↪  r_index=0, v_index=1)
```

Inspection

```
# Take the 0th batch of particles
c1Maxwell_batch = c1Maxwell.batches[0]['H ions']
# Take the initial positions and velocities of the particles
c1Maxwell_positions = []
c1Maxwell_velocities = []
for particle in c1Maxwell_batch.particles:
c1Maxwell_positions.append(particle.r)
c1Maxwell_velocities.append(particle.v)
# Let's now look at the velocities
c1Maxwell_velocities
# We need to check if they are really Maxwellian distributed
# Get the speeds
c1Maxwell_speeds = np.sqrt( [ (c1Maxwell_velocities[i][0] ** 2) +
↪  (c1Maxwell_velocities[i][1] ** 2) + (c1Maxwell_velocities[i][2] ** 2) for i in
↪  range(len(c1Maxwell_velocities)) ] )
c1Maxwell_meanspeed = np.sum(c1Maxwell_speeds) / c1Maxwell_speeds.size
```

We get an average speed of the distribution to be: 14202.7645 ms^{-1}

We see that the mean average speed of the sampled distribution and that expected from the analytic expression are close; in fact within 2.0431% error.

Initialization

- Hydrogen molecule sampling
- We get the same temperature for this distribution but the relative molecular mass is doubled

```
# Create 100 particles based on the sampled velocities of H2 gas
# We consider just for this test case that a Hydrogen molecules to be sampled with a Maxwellian distribution
# We create a new batch on the same Run instance
c1Maxwell.create_batch_with_file_initialization('H2', constants.constants['e'][0], 2 * constants.constants['m_H'][0] * constants.constants['amu'][0], 100, 100, 'H2 gas', r_index=0, v_index=2)
```

Inspection

```
# Do the same as before for the second batch
c1Maxwell_batch2 = c1Maxwell.batches[1]['H2 gas']
c1Maxwell_positions_batch2 = []
c1Maxwell_velocities_batch2 = []
for particle in c1Maxwell_batch2.particles:
c1Maxwell_positions_batch2.append(particle.r)
c1Maxwell_velocities_batch2.append(particle.v)
c1Maxwell_speeds_batch2 = np.sqrt( [ (c1Maxwell_velocities_batch2[i][0] ** 2) +
↪  (c1Maxwell_velocities_batch2[i][1] ** 2) + (c1Maxwell_velocities_batch2[i][2] ** 2)
↪  for i in range(len(c1Maxwell_velocities_batch2)) ] )
c1Maxwell_meanspeed_batch2 = np.sum(c1Maxwell_speeds_batch2) /
↪  c1Maxwell_speeds_batch2.size
```

- We get an average speed of the distribution to be: 10149.754 ms^{-1} .
- From the analytic calculation for the Hydrogen molecule particle distribution, we get

$$\langle v \rangle = \frac{2}{\sqrt{\pi}} \sqrt{\frac{2 \cdot 8.31446261815324 \text{ J K}^{-1} \text{ mol}^{-1} \ 10000 \text{ K}}{2 \times 1.008 \times 10^{-3} \text{ kg mol}^{-1}}} = 10248.06534135222 \text{ ms}^{-1}$$

- The mean average speed of the sampled distribution and that from the analytic expression are close for this distribution within 0.968% error.

1.2 Update

- We recall the Boris Algorithm, that we use to update the particles in the plasma simulation.

$$\begin{aligned} \boldsymbol{v}^{-} &= \boldsymbol{v}_k + q'\mathbf{E}_k \\ \boldsymbol{v}^{+} &= \boldsymbol{v}^{-} + 2q'\left(\boldsymbol{v}^{-} \times \mathbf{B}_k\right) \\ \boldsymbol{v}_{k+1} &= \boldsymbol{v}^{+} + q'\mathbf{E}_k \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \Delta t \ \boldsymbol{v}_{k+1} \end{aligned} \tag{3}$$

where q 0 = q m /Δt 2 . To verify that the simulation works as we expect it to, we perform a calculation and compare it to the output of an algorithm.

Initialization

```
# c2Boris means checking the Boris update
c2Boris = Run()
#Create 10 particles
c2Boris.create_batch_with_file_initialization('H+', constants.constants['e'][0],
↪  constants.constants['m_H'][0] * constants.constants['amu'][0], 100, 10, 'H ions',
↪  r_index=0, v_index=1)
```

Update input data

```

c2Boris_index_update = 0 # Update the first batch in this Run instance run_Boris_check
c2Boris_particle_track_indices = [i for i in range(10)] # Track all 10 particles
c2Boris_dT = 10**(-6) # 1 microseconds
c2Boris_stepT = 10**(-7) # 0.1 microseconds time step
c2Boris_E0 = 1000 # say 1000 Volts (voltage) per meter (size of chamber)
c2Boris_Edirn = [1,0,0] #in the x-direction [1,0,0]
c2Boris_B0 = 10 * (10**(-3)) # Meant to say 10 mT
c2Boris_Bdirn = [0,1,0] #in the y-direction [0,1,0]
c2Boris_argsE = [element * c2Boris_E0 for element in c2Boris_Edirn] # currently the
↪ uniform_E_field configuration is used
c2Boris_argsB = [element * c2Boris_B0 for element in c2Boris_Bdirn]# currently the
↪ uniform_B_field configuration is used
```

```
c2Boris_positions_and_velocities =
↪ c2Boris.update_batch_with_unchanging_fields(c2Boris_index_update, c2Boris_dT,
↪ c2Boris_stepT, c2Boris_argsE, c2Boris_argsB, c2Boris_particle_track_indices)

#Let's inspect the positions and velocities of the particles at index 1 and 7
c2Boris_p1 = c2Boris_positions_and_velocities[1]
c2Boris_p7 = c2Boris_positions_and_velocities[7]

#Let's take the positions and velocities of the particle 1 after the 3rd and the 4th
↪ time steps.
c2Boris_p134_p = [c2Boris_p1[3][1], c2Boris_p1[4][1]]
c2Boris_p134_v = [c2Boris_p1[3][2], c2Boris_p1[4][2]]

#Similary for particle 7
c2Boris_p734_p = [c2Boris_p7[3][1], c2Boris_p7[4][1]]
c2Boris_p734_v = [c2Boris_p7[3][2], c2Boris_p7[4][2]]
```

- We take the following input data:

100 particles of Hydrogen

Electric field of 1000 Vm-1 in x direction [1,0,0]

Magnetic field of 10 mT in y direction [0,1,0]

Duration of simulation 1μs

Duration of 1 step of update = 0.1 μs

$$\boldsymbol{v}_4 = \boldsymbol{v}^+ + q'\mathbf{E} = \begin{bmatrix} 44123.88704013 \\ -5029.491038 \\ 8798.39924387 \end{bmatrix} + 4.78597877761177 \begin{bmatrix} 1000 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 48909.86581774 \\ -5029.491038 \\ 8798.39924387 \end{bmatrix}$$

$$\boldsymbol{x}_4 = \boldsymbol{x}_3 + \Delta t \;\; \boldsymbol{v}_4 = \begin{bmatrix} -4.89851127 \times 10^{-01} \\ -2.01179642 \times 10^{-03} \\ 1.70051838 \times 10^{-04} \end{bmatrix} + 10^{-7} \begin{bmatrix} 48909.86581774 \\ -5029.491038 \\ 8798.39924387 \end{bmatrix} = \begin{bmatrix} -0.48496014 \\ -0.00251475 \\ 0.00104989 \end{bmatrix}$$

- We see that the values for x3, x4, v3 and v4 for particle 1 are close (the latter digits differ due to the differing precision of calculations); i.e. the same when done by hand and in the program.

$$\boldsymbol{v}_4 = \boldsymbol{v}^+ + q'\mathbf{E} = \begin{bmatrix} 42349.90252151 \\ -8670.854777 \\ 18096.17636737 \end{bmatrix} + 4.78597877761177 \begin{bmatrix} 1000 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 47135.88129912 \\ -8670.854777 \\ 18096.17636737 \end{bmatrix}$$

$$\boldsymbol{x}_4 = \boldsymbol{x}_3 + \Delta t \;\; \boldsymbol{v}_4 = \begin{bmatrix} -0.48966698 \\ -0.00346834 \\ 0.00388755 \end{bmatrix} + 10^{-7} \begin{bmatrix} 47135.88129912 \\ -8670.854777 \\ 18096.17636737 \end{bmatrix} = \begin{bmatrix} -0.48495339 \\ -0.00433543 \\ 0.00569717 \end{bmatrix}$$

Update

- We see that the values for x3, x4, v3 and v4 for particle 1 are close (the latter digits differ due to the differing precision of calculations); i.e. the same when done by hand and in the program.
- Similarly, for particle 7. After the 3rd step of the update, the velocity of the particle was:

```
[38895.85871094631, -8670.854777, 13914.969423620274] ms-1
```

- and after the 4th update, it changed to:

```
[47135.881299118584, -8670.854777, 18096.176367366777] ms-1
```

- Its position after the 3rd step of the update was

```
[-0.4896669752432719, -0.0034683419108, 0.0038875496903932644]
```

- which was updated after the 4th step of the update to:

```
[-0.48495338711336006, -0.0043354273885, 0.0056971673271299424]
```

Plasma Stream

We now create our first plasma system, where we can change certain parameters of the fields; so as to simulate controlling plasma in a chamber by changing the electric and magnetic fields as required.

- We take the following input data:
- 100 ions of Hydrogen
  speeds are Maxwellian sampled with 10,000 K Temperature
- velocity directions are uniform randomly sampled
- positions are all sampled such that particles start at [-0.5, 0, 0]
- (a chamber 1m x 1m x 1m with extreme points [-0.5, -0.5, -0.5] and [0.5, 0.5, 0.5] considered)
- Here we consider constant magnetic field of 10 mT along the y-axis [0,1,0] and changing electric field configurations of [0, 0, 1, -1, 2, -2, 3, -3, 4, -4, 5, -5] 1000 Vm−1 along the x-axis [1,0,0]. For a time duration of 0.1 ms we update the batch of particles under different configurations of the electric field, using time steps of 0.001 ms

```
#Create a constants object instance to access the constants from constants.ipynb file
constants = Constants()
# Create a run object instance
s1 = Run()
# Create 100 Hydrogen ions whose:
# speeds are Maxwellian sampled, velocity directions are uniform randomly sampled
# positions are all sampled such that particles start at [-0.5, 0, 0]
# a chamber 1m x 1m x 1m with extreme points [-0.5, -0.5, -0.5] and [0.5, 0.5, 0.5]
↪  considered
s1.create_batch_with_file_initialization('H+', constants.constants['e'][0],
↪  constants.constants['m_H'][0] * constants.constants['amu'][0], 100, 100, 'H ions',
↪  r_index=0, v_index=1)
```

```
# Take the first batch in this run object
s1_batch1 = s1.batches[0]['H ions']

# Let's consider Electric field being flipped in direction, and taken up a few scales
E_scale = [0, 0, 1, -1, 2, -2, 3, -3, 4, -4, 5, -5]
# Let's consider a case with constant Magnetic field
B_scale = [1 for i in range(len(E_scale))]

s1_index_update = 0 # Update the first batch in this Run instance
s1_particle_track_indices = [i for i in range(100)] # Track all 100 particles
s1_dT = 10**(-7) # 0.1 microseconds
s1_stepT = 10**(-9) # 0.001 microseconds time step
s1_E0 = 1000 # say 1000 Volts (voltage) per meter (size of chamber)
s1_Edirn = [1,0,0] #in the x-direction [1,0,0]
s1_B0 = 10 * (10**(-3)) # Meant to say 10 mT
s1_Bdirn = [0,1,0] #in the y-direction [0,1,0]

s1_argsE = [element * s1_E0 for element in s1_Edirn] # currently the uniform_E_field configuration is used
s1_argsB = [element * s1_B0 for element in s1_Bdirn] # currently the uniform_B_field configuration is used

s1_batch_ps_and_vs = dict()

for i in range(len(E_scale)):
        desc = 'E_scale = ' + str(E_scale[i]) + ' ' + 'B_scale = ' + str(B_scale[i])
        s1_batch1_ps_and_vs_once = s1.update_batch_with_unchanging_fields(s1_index_update, s1_dT, s1_stepT, [elem *E_scale[i] for elem in s1_argsE], [elem *E_scale[i] for elem in s1_argsB] * B_scale[i], s1_particle_track_indices)
        s1_batch_ps_and_vs[desc] = s1_batch1_ps_and_vs_once
```

- **Now we need to extract and arrange the positions and velocities of the particles during the update history and plot them. Let's first look at the particle at index 0.**
  Let's first extract the position and velocities of particle zero when the electric field was scaled by +1 and -1 and plot the positions.

```
s1_descE_is_1 = 'E_scale = ' + str(E_scale[2]) + ' ' + 'B_scale = ' + str(B_scale[2])
s1_descE_is_n1 = 'E_scale = ' + str(E_scale[3]) + ' ' + 'B_scale = ' + str(B_scale[3])
# Extract update histories for two field configs
s1_histories_E1 = s1_batch_ps_and_vs[s1_descE_is_1]
s1_histories_nE1 = s1_batch_ps_and_vs[s1_descE_is_n1]

# Extract information on 0th particle's update history in both cases
s1_descE_is_1_p0 = s1_histories_E1[0]
s1_descE_is_n1_p0 =  s1_histories_nE1[0]

#Get positions and velocities of the particle's update history
s1_descE_is_1_p0_ps = []
s1_descE_is_1_p0_vs = []

for i in range(len(s1_descE_is_1_p0)):
        s1_descE_is_1_p0_ps.append(s1_descE_is_1_p0[i][1])
        s1_descE_is_1_p0_vs.append(s1_descE_is_1_p0[i][2])

        s1_descE_is_n1_p0_ps = []
        s1_descE_is_n1_p0_vs = []

for i in range(len(s1_descE_is_n1_p0)):
        s1_descE_is_n1_p0_ps.append(s1_descE_is_n1_p0[i][1])
        s1_descE_is_n1_p0_vs.append(s1_descE_is_n1_p0[i][2])
```

- Now we plot the positions of the particle at index 0 when the electric field was scaled by +1 and -1.

```
# Plot the position update history of particle 0 when the electric field is scaled by 1
s1_descE_is_1_p0_ps_fig = plt.figure()
s1_descE_is_1_p0_ps_ax = plt.axes(projection='3d')
s1_descE_is_1_p0_ps_ax.view_init(50, -20)

# Data for three-dimensional scattered points
# for position update history of particle 0 when the Electric field was scaled by 1
s1_descE_is_1_p0_ps_zdata = [elem[2] for elem in s1_descE_is_1_p0_ps]
s1_descE_is_1_p0_ps_xdata = [elem[0] for elem in s1_descE_is_1_p0_ps]
s1_descE_is_1_p0_ps_ydata = [elem[1] for elem in s1_descE_is_1_p0_ps]
s1_descE_is_1_p0_ps_ax.scatter3D(s1_descE_is_1_p0_ps_xdata, s1_descE_is_1_p0_ps_ydata,
↪  s1_descE_is_1_p0_ps_zdata,\
c=s1_descE_is_1_p0_ps_zdata, cmap='Greens');
plt.savefig('EplusPs', dpi='figure', format='png')

# Plot the position update history of particle 0 when the electric field is scaled by -1
# We see an opposite curve to the previous figure
s1_descE_is_n1_p0_ps_fig = plt.figure()
s1_descE_is_n1_p0_ps_ax = plt.axes(projection='3d')
s1_descE_is_n1_p0_ps_ax.view_init(50, -20)

# Data for three-dimensional scattered points
# for position update history of particle 0 when the Electric field was scaled by -1
s1_descE_is_n1_p0_ps_zdata = [elem[2] for elem in s1_descE_is_n1_p0_ps]
s1_descE_is_n1_p0_ps_xdata = [elem[0] for elem in s1_descE_is_n1_p0_ps]
s1_descE_is_n1_p0_ps_ydata = [elem[1] for elem in s1_descE_is_n1_p0_ps]
s1_descE_is_n1_p0_ps_ax.scatter3D(s1_descE_is_n1_p0_ps_xdata,
↪  s1_descE_is_n1_p0_ps_ydata, \
s1_descE_is_n1_p0_ps_zdata, c=s1_descE_is_n1_p0_ps_zdata, cmap='Greens');
plt.savefig('EminusPs', dpi='figure', format='png')
```

- **The plots output are as following. All numbers are in meters. In 3-dimensional plots, the vertical axis represents the z-axis, the axis into the page of the plane represents the x-axis and the axis along the lines the page represents the y-axis. In 1 dimensional plots, the vertical axis represents the value being plotted and the horizontal axis represents the steps of iteration.**

Particle trajectory in opposite Electric Fields

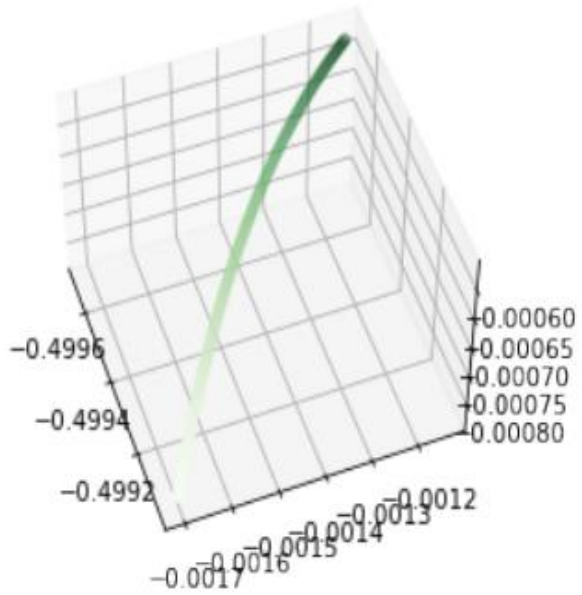


Figure 1: Electric field scaled by 1

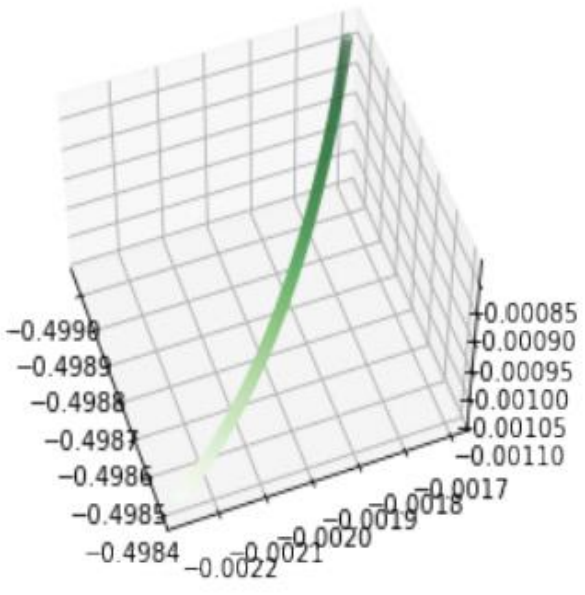


Figure 2: Electric field scaled by -1

2.2 Plotting particle update history

```
# take for a field configuration
s1_allFieldkeys = list(s1_batch_ps_and_vs.keys())
s1_allfield_p0_ps = []
s1_allfield_p0_vs = []
for akey in s1_allFieldkeys:
s1_histories = s1_batch_ps_and_vs[akey]

#Take particle 0
s1_p0 = s1_histories[0]

#Take ps and vs
for i in range(len(s1_p0)):
        s1_allfield_p0_ps.append(s1_p0[i][1])
        s1_allfield_p0_vs.append(s1_p0[i][2])
        s1_allfield_p0_ps = np.array(s1_allfield_p0_ps)
        s1_allfield_p0_vs = np.array(s1_allfield_p0_vs)

#Plot the position
s1_allfield_p0_ps_fig = plt.figure()
s1_allfield_p0_ps_ax = plt.axes(projection='3d')
s1_allfield_p0_ps_ax.view_init(20, -50)

# Data for three-dimensional scattered points
# for position update history of particle 0 during all field configurations
s1_allfield_p0_ps_zdata = [elem[2] for elem in s1_allfield_p0_ps]
s1_allfield_p0_ps_xdata = [elem[0] for elem in s1_allfield_p0_ps]
s1_allfield_p0_ps_ydata = [elem[1] for elem in s1_allfield_p0_ps]
s1_allfield_p0_ps_ax.scatter3D(s1_allfield_p0_ps_xdata, s1_allfield_p0_ps_ydata,
↪  s1_allfield_p0_ps_zdata,\
c=s1_allfield_p0_ps_zdata, cmap='Greens');
plt.savefig('ps1', dpi='figure', format='png')
```

```
# Plot x position
s1_allfield_p0_ps_x_fig = plt.figure()
s1_allfield_p0_ps_x_ax = plt.axes()
s1_allfield_p0_ps_x_ax.scatter(np.arange(len(s1_allfield_p0_ps_xdata)),
↪  s1_allfield_p0_ps_xdata);
plt.savefig('psx1', dpi='figure', format='png')

# Plot y position
s1_allfield_p0_ps_y_fig = plt.figure()
s1_allfield_p0_ps_y_ax = plt.axes()
s1_allfield_p0_ps_y_ax.scatter(np.arange(len(s1_allfield_p0_ps_ydata)),
↪  s1_allfield_p0_ps_ydata);
plt.savefig('psy1', dpi='figure', format='png')

# Plot z position
s1_allfield_p0_ps_z_fig = plt.figure()
s1_allfield_p0_ps_z_ax = plt.axes()
s1_allfield_p0_ps_z_ax.scatter(np.arange(len(s1_allfield_p0_ps_zdata)),
↪  s1_allfield_p0_ps_zdata);
plt.savefig('psz1', dpi='figure', format='png')
```

- We also plot the x, y and z components of position during the evolution, against the update steps.

Position and components plot

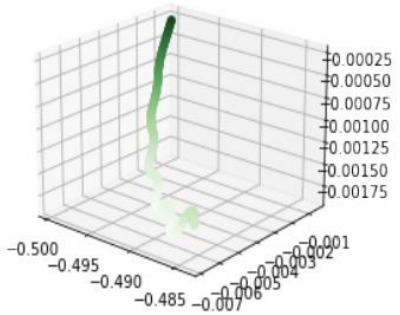


Figure 3: position

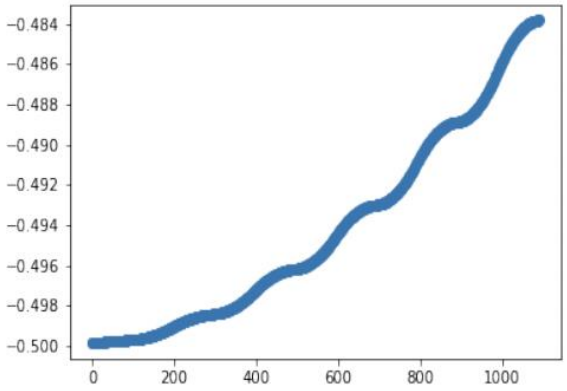


Figure 4: x-component of position

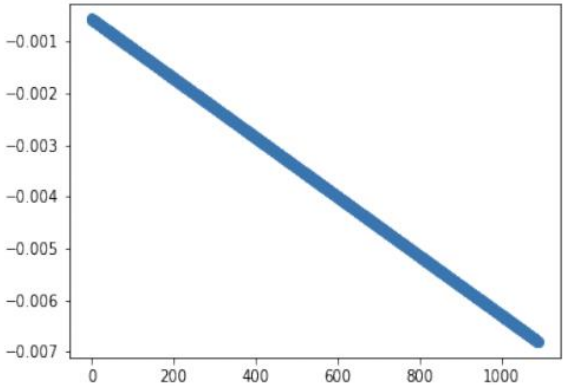


Figure 5: y-component of position

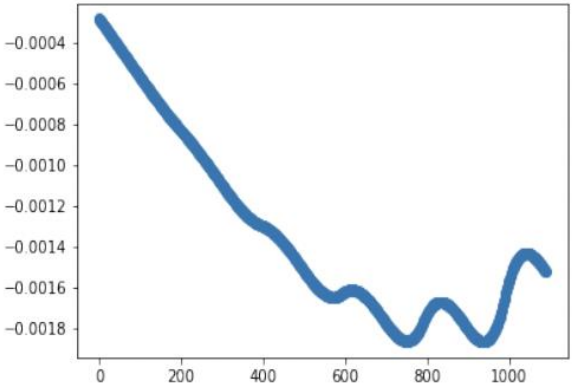


Figure 6: z-component of position

Animation for plots

```
def plot_animation_3d(positions):
        '''
        This function can plot both positions and velocities
        '''

        FRAMES = np.shape(positions)[0]
        # Here positions has shape (number of particles, 3) where each entry is a
        ↪  position which is an array of x,y,z coordinates
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')

        def init():
                ax.view_init(elev=10., azim=0)
                ax.set_xlabel('x')
                ax.set_ylabel('y')
                ax.set_zlabel('z')

        # animation function.  This is called sequentially
        def animate(i):
                current_index = int(positions.shape[0] / FRAMES * i)
                ax.cla()
                ax.view_init(elev=10., azim=i)
                ax.set_xlabel('x')
                ax.set_ylabel('y')
                ax.set_zlabel('z')
                # For line plot uncomment the following line
                # ax.plot3D(positions[:current_index, 0], positions[:current_index, 1],
                ↪  positions[:current_index, 2])
                ax.scatter3D(positions[:current_index, 0], positions[:current_index, 1],
                ↪  positions[:current_index, 2])

                # call the animator.
                anim = animation.FuncAnimation(fig, animate, init_func=init,
                ↪  frames=FRAMES, interval=100)

                return anim

```

Animations have been generated for all plots and they help us understand these update histories better. Animations were generated using the following parts of the program.

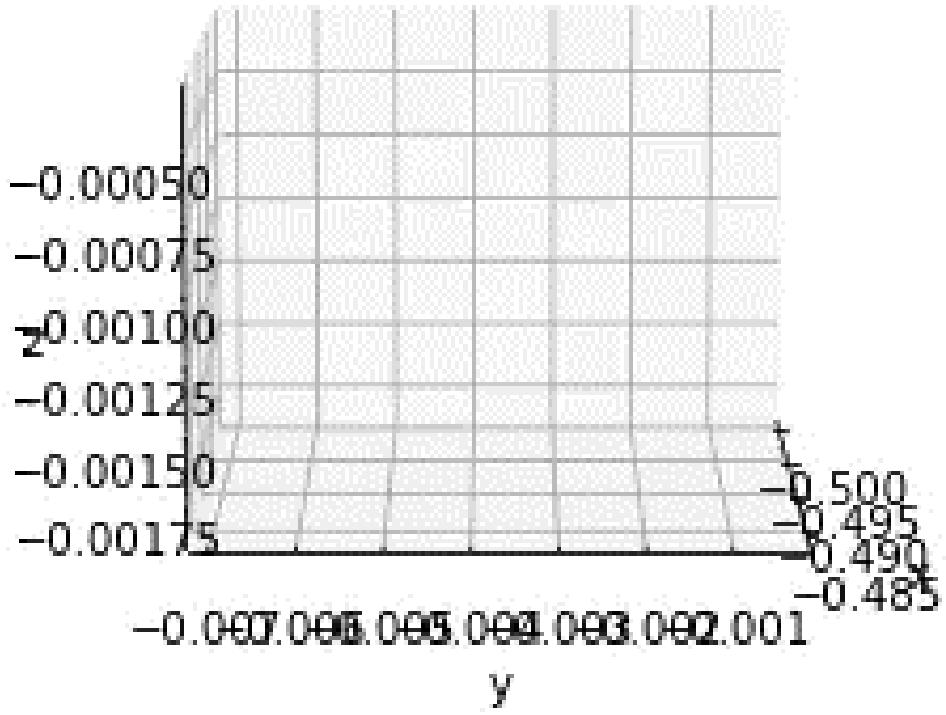
```
34  def plot_animation_1d(positions, include):
35          '''
36          This function can plot both positions and velocities
37          include can be 0, 1 or 2.
38          if include = 2, this means plot the z data of the array
39          '''
40
41          FRAMES = np.shape(positions)[0]
42          # Here positions has shape (number of particles, 3) where each entry is a
            ↪  position which is an array of x,y,z coordinates
43          fig = plt.figure()
44          ax = fig.add_subplot(111)
45
46          def init():
47                  ax.set_xlabel('step')
48                  ax.set_ylabel(chr(include + 120))
49
50          # animation function.  This is called sequentially
51          def animate(i):
52                  current_index = int(positions.shape[0] / FRAMES * i)
53                  ax.cla()
54                  ax.set_xlabel('step')
55                  ax.set_ylabel(chr(include + 120))
56                  # For line plot uncomment the following line
57                  # ax.plot3D(positions[:current_index, 0], positions[:current_index, 1],
                    ↪  positions[:current_index, 2])
58                  ax.scatter(np.arange(len(positions))[:current_index],
                    ↪  positions[:current_index, include])
59
60                  # call the animator.
61                  anim = animation.FuncAnimation(fig, animate, init_func=init,
                    ↪  frames=FRAMES, interval=100)
62
63                  return anim
```

Running the animation generation.

```
# animation for the position of the particle
s1_allfield_p0_ps_anim = plot_animation_3d(s1_allfield_p0_ps)
display_animation(s1_allfield_p0_ps_anim)
s1_allfield_p0_ps_anim.save(r'ps1.mp4')

# x positions
s1_allfield_p0_ps_x_anim = plot_animation_1d(s1_allfield_p0_ps, include=0)
display_animation(s1_allfield_p0_ps_x_anim)
s1_allfield_p0_ps_x_anim.save(r'psx1.mp4')

# y positions
s1_allfield_p0_ps_y_anim = plot_animation_1d(s1_allfield_p0_ps, include=1)
display_animation(s1_allfield_p0_ps_y_anim)
s1_allfield_p0_ps_y_anim.save(r'psy1.mp4')

# z positions
s1_allfield_p0_ps_z_anim = plot_animation_1d(s1_allfield_p0_ps, include=2)
display_animation(s1_allfield_p0_ps_z_anim)
s1_allfield_p0_ps_z_anim.save(r'psz1.mp4')

# Plot the velocity
s1_allfield_p0_vs_anim = plot_animation_3d(s1_allfield_p0_vs)
display_animation(s1_allfield_p0_vs_anim)
s1_allfield_p0_vs_anim.save(r'vs1.mp4')

# Plot x velocity
s1_allfield_p0_vs_x_anim = plot_animation_1d(s1_allfield_p0_vs, include=0)
display_animation(s1_allfield_p0_vs_x_anim)
s1_allfield_p0_vs_x_anim.save(r'vsx1.mp4')

# Plot y velocity
s1_allfield_p0_vs_y_anim = plot_animation_1d(s1_allfield_p0_vs, include=1)
display_animation(s1_allfield_p0_vs_y_anim)
s1_allfield_p0_vs_y_anim.save(r'vsy1.mp4')

# Plot z velocity
s1_allfield_p0_vs_z_anim = plot_animation_1d(s1_allfield_p0_vs, include=2)
display_animation(s1_allfield_p0_vs_z_anim)
s1_allfield_p0_vs_z_anim.save(r'vsz1.mp4')
```

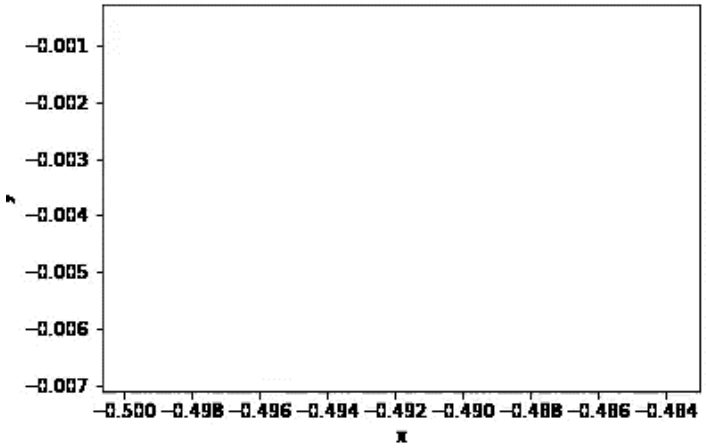
```
VIDEO_TAG = """<video controls>
<source src="data:video/x-m4v;base64,{0}" type="video/mp4">
Your browser does not support the video tag.
</video>"""

def anim_to_html(anim):
        if not hasattr(anim, '_encoded_video'):
                f = NamedTemporaryFile(suffix='.mp4', delete=False)
                anim.save(f.name, fps=20, extra_args=['-vcodec', 'libx264', '-pix_fmt',
                ↪  'yuv420p'])
                f.flush()
                video = open(f.name, "rb").read()
                f.close()
                anim._encoded_video = base64.b64encode(video).decode('utf-8')

        return VIDEO_TAG.format(anim._encoded_video)

def display_animation(anim):
        plt.close(anim._fig)
        return HTML(anim_to_html(anim))
```

Position animation

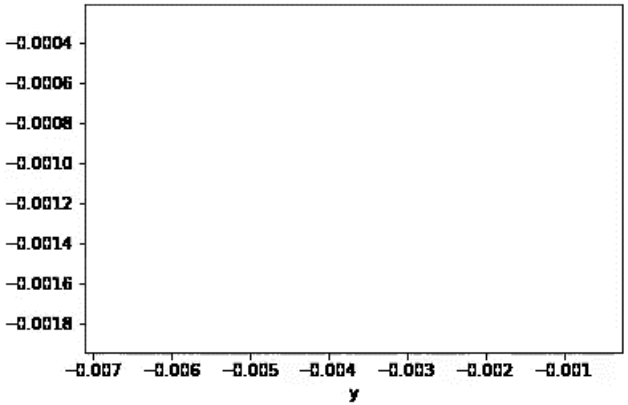


Position animations

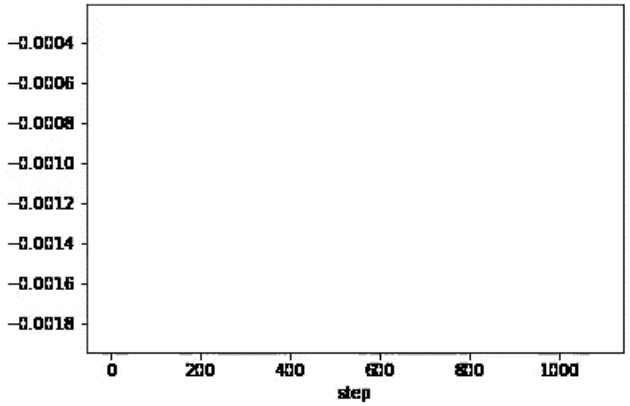
Position in x component



Position in y component



Position in z component



2.3 Plotting velocity update history

```
# Plot the velocity
s1_allfield_p0_vs_fig = plt.figure()
s1_allfield_p0_vs_ax = plt.axes(projection='3d')
s1_allfield_p0_vs_ax.view_init(20, -80)

# Data for three-dimensional scattered points
# for position update history of particle 0 during all field configurations
s1_allfield_p0_vs_zdata = [elem[2] for elem in s1_allfield_p0_vs] # Animate this plot as
↪ well.
s1_allfield_p0_vs_xdata = [elem[0] for elem in s1_allfield_p0_vs]
s1_allfield_p0_vs_ydata = [elem[1] for elem in s1_allfield_p0_vs]
```

```
s1_allfield_p0_vs_ax.scatter3D(s1_allfield_p0_vs_xdata, s1_allfield_p0_vs_ydata,
↪ s1_allfield_p0_vs_zdata,\
c=s1_allfield_p0_vs_zdata, cmap='Greens');
plt.savefig('vs1', dpi='figure', format='png')

# Plot x velocity
s1_allfield_p0_vs_x_fig = plt.figure()
s1_allfield_p0_vs_x_ax = plt.axes()
s1_allfield_p0_vs_x_ax.scatter(np.arange(len(s1_allfield_p0_vs_xdata)),
↪ s1_allfield_p0_vs_xdata);
plt.savefig('vsx1', dpi='figure', format='png')

# Plot y velocity
s1_allfield_p0_vs_y_fig = plt.figure()
s1_allfield_p0_vs_y_ax = plt.axes()
s1_allfield_p0_vs_y_ax.scatter(np.arange(len(s1_allfield_p0_vs_ydata)),
↪ s1_allfield_p0_vs_ydata);
plt.savefig('vsy1', dpi='figure', format='png')

# Plot z velocity
s1_allfield_p0_vs_z_fig = plt.figure()
s1_allfield_p0_vs_z_ax = plt.axes()
s1_allfield_p0_vs_z_ax.scatter(np.arange(len(s1_allfield_p0_vs_zdata)),
↪ s1_allfield_p0_vs_zdata);
plt.savefig('vsz1', dpi='figure', format='png')
```

We also plot the velocity and components of velocity of the particle.

Velocity and components plot

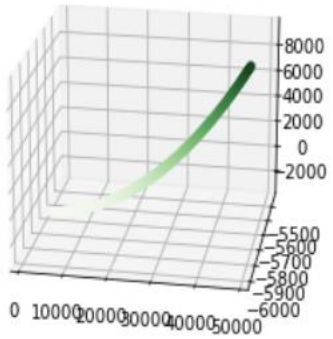


Figure 7: velocity

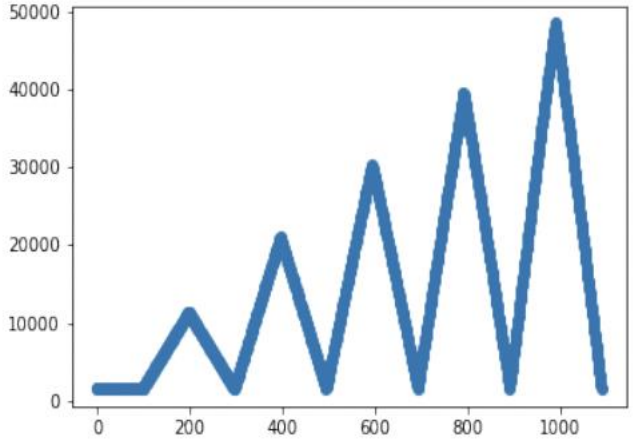


Figure 8: x-component of velocity

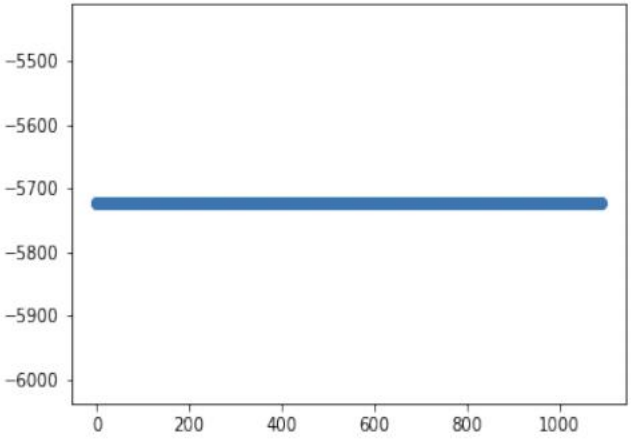


Figure 9: y-component of velocity

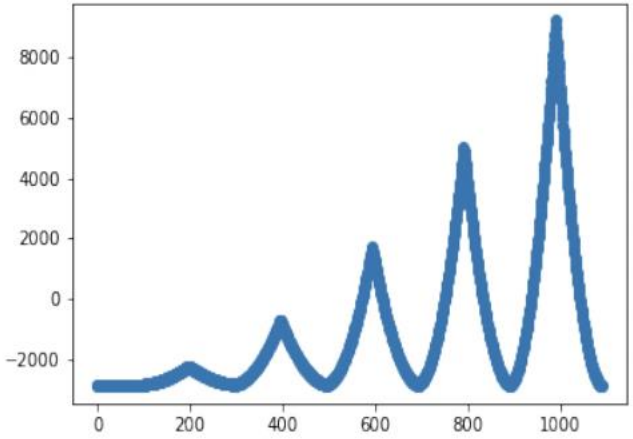
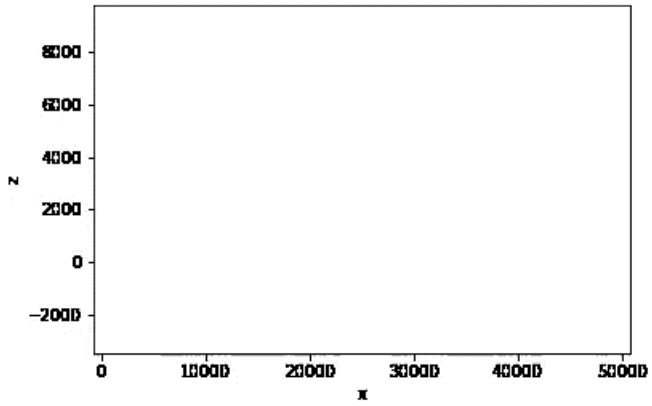


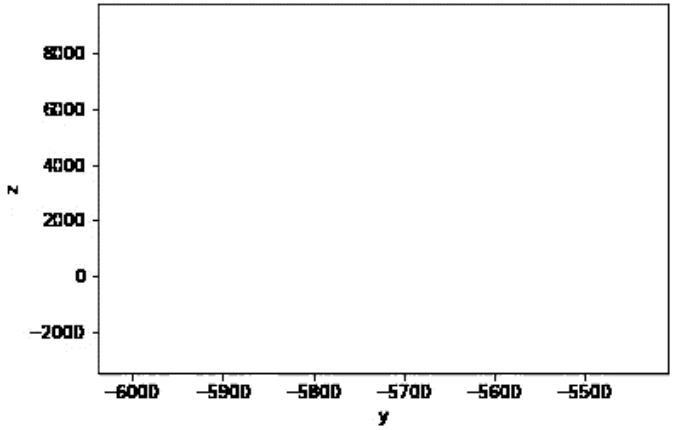
Figure 10: z-component of velocity

Velocity animations

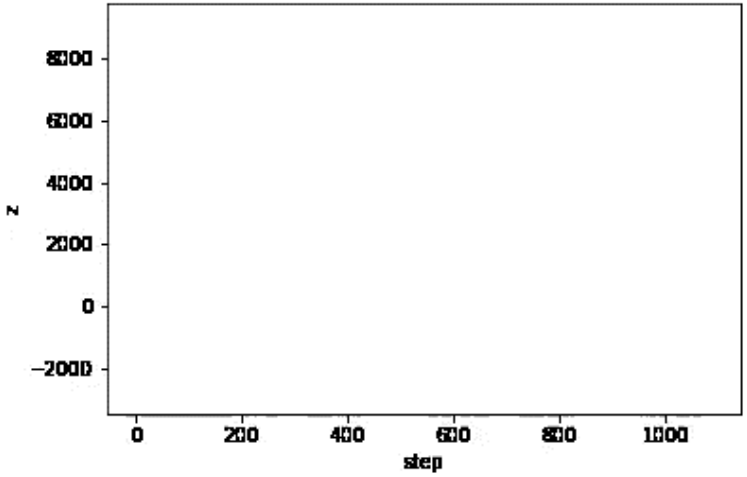
Velocity in x component



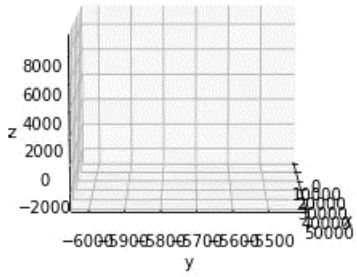
Velocity in y component



Velocity in z component



Velocity in 3D



2.3 Multiparticle update plots

We can look at the position and velocity update histories of many particles. We pick 10 particles, the particles at indices [0, 10, 20, 30, 40, 50, 60, 70, 80, 90] to plot.

```
s1_particles = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90] # take particles at these indices
s1_allFieldkeys = list(s1_batch_ps_and_vs.keys())
s1_allfield_10p_ps = []
s1_allfield_10p_vs = []

for aparticle in s1_particles:
s1_allfield_ap_ps = []
s1_allfield_ap_vs = []

# Same procedure for as a single particle
for akey in s1_allFieldkeys:
        s1_histories = s1_batch_ps_and_vs[akey]

#Take aparticle
s1_ap = s1_histories[aparticle]

#Take ps and vs
for i in range(len(s1_p0)):
        s1_allfield_ap_ps.append(s1_ap[i][1])
        s1_allfield_ap_vs.append(s1_ap[i][2])
        s1_allfield_ap_ps = np.array(s1_allfield_ap_ps)
        s1_allfield_ap_vs = np.array(s1_allfield_ap_vs)

```

```
        s1_allfield_10p_ps.append(s1_allfield_ap_ps)
        s1_allfield_10p_vs.append(s1_allfield_ap_vs)

s1_allfield_10p_ps = np.array(s1_allfield_10p_ps)
s1_allfield_10p_vs = np.array(s1_allfield_10p_vs)

# Plot the positions
s1_allfield_10p_ps_fig = plt.figure()
s1_allfield_10p_ps_ax = plt.axes(projection='3d')
s1_allfield_10p_ps_ax.view_init(20, -80)

# Data for three-dimensional scattered points
for i in range(len(s1_particles)):
        s1_allfield_ap_ps_zdata = [elem[2] for elem in s1_allfield_10p_ps[i]]
        s1_allfield_ap_ps_xdata = [elem[0] for elem in s1_allfield_10p_ps[i]]
        s1_allfield_ap_ps_ydata = [elem[1] for elem in s1_allfield_10p_ps[i]]
        s1_allfield_10p_ps_ax.scatter3D(s1_allfield_ap_ps_xdata,
        ↪  s1_allfield_ap_ps_ydata, s1_allfield_ap_ps_zdata,
        ↪  c=s1_allfield_p0_ps_zdata);
plt.savefig('multips1', dpi='figure', format='png')

# Plot x positions
s1_allfield_10p_ps_x_fig = plt.figure()
s1_allfield_10p_ps_x_ax = plt.axes()
for i in range(len(s1_particles)):
        s1_allfield_ap_ps_xdata = [elem[0] for elem in s1_allfield_10p_ps[i]]
        s1_allfield_10p_ps_x_ax.scatter(np.arange(len(s1_allfield_ap_ps_xdata)),
        ↪  s1_allfield_ap_ps_xdata);
plt.savefig('multipsx1', dpi='figure', format='png')
```

```
50
51  # Plot y positions
52  s1_allfield_10p_ps_y_fig = plt.figure()
53  s1_allfield_10p_ps_y_ax = plt.axes()
54  for i in range(len(s1_particles)):
55          s1_allfield_ap_ps_ydata = [elem[1] for elem in s1_allfield_10p_ps[i]]
56          s1_allfield_10p_ps_y_ax.scatter(np.arange(len(s1_allfield_ap_ps_ydata)),
            ↪  s1_allfield_ap_ps_ydata);
57  plt.savefig('multipsy1', dpi='figure', format='png')
58
59  # Plot z positions
60  s1_allfield_10p_ps_z_fig = plt.figure()
61  s1_allfield_10p_ps_z_ax = plt.axes()
62  for i in range(len(s1_particles)):
63          s1_allfield_ap_ps_zdata = [elem[2] for elem in s1_allfield_10p_ps[i]].
64          s1_allfield_10p_ps_z_ax.scatter(np.arange(len(s1_allfield_ap_ps_zdata)),
            ↪  s1_allfield_ap_ps_zdata);
65  plt.savefig('multipsz1', dpi='figure', format='png')
```

Plots of the positions:

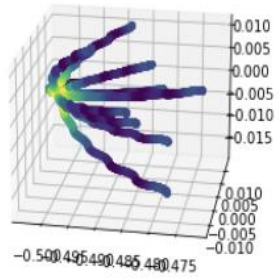


Figure 11: positions

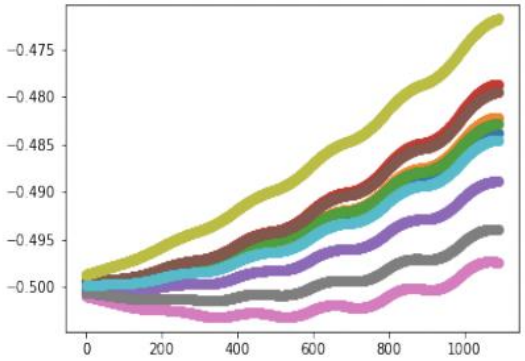


Figure 12: x-component of positions

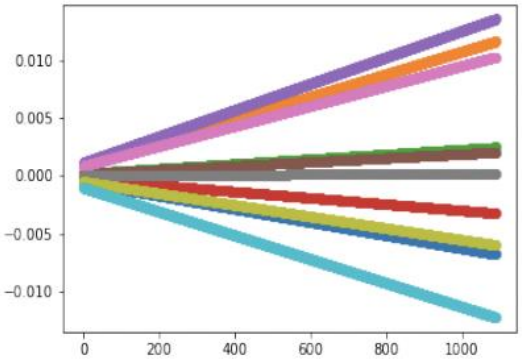


Figure 13: y-component of positions

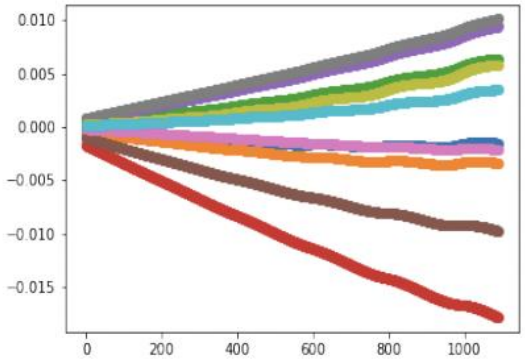


Figure 14: z-component of positions

```
def multiplot_animation_3d(positions):
        '''
        Here each element of positions is data for 1 particle that one would give as input to
        plot_animation_3d function, i.e. position or velocity update history of 1 particle
```

```

        This function can plot both positions and velocities
        '''

        #positions = np.array(np.array([xdata, ydata, zdata]))
        FRAMES = np.shape(positions)[1]
        # Here positions has shape (10, 1089, 3)
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')

        def init():
                ax.view_init(elev=20., azim=0)
                ax.set_xlabel('x')
                ax.set_ylabel('y')
                ax.set_zlabel('z')

        # animation function.  This is called sequentially
        def animate(i):
                current_index = int(positions.shape[1] / FRAMES * i)
                ax.cla()
                ax.view_init(elev=20., azim=i)
                ax.set_xlabel('x')
                ax.set_ylabel('y')
                ax.set_zlabel('z')
```

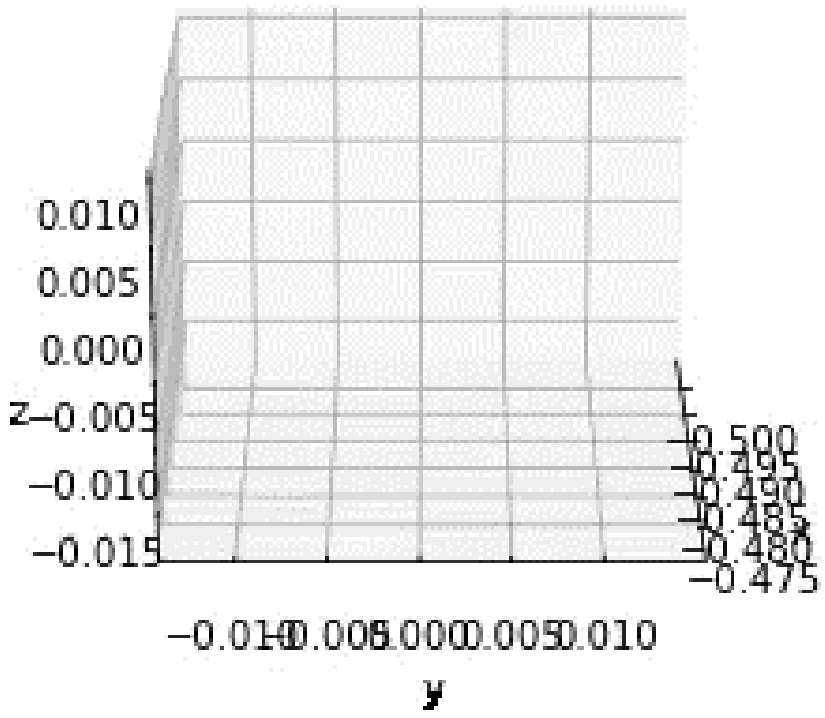
```
                # For line plot uncomment the following line
                # ax.plot3D(positions[:current_index, 0], positions[:current_index, 1],
                ↪  positions[:current_index, 2])
                for position in positions:
                ax.scatter3D(position[:current_index, 0], position[:current_index, 1],
                ↪  position[:current_index, 2])

                # call the animator.
                anim = animation.FuncAnimation(fig, animate, init_func=init,
                ↪  frames=FRAMES, interval=100)

                return anim

def multiplot_animation_1d(positions, include):
        '''Here each element of positions is data for 1 particle that one would give as
        ↪  input to
        plot_animation_3d function, i.e. position or velocity update history of 1
↪  particle

        This function can plot both positions and velocities

        include can be 0, 1 or 2.
        if include = 2, this means plot the z data of the array
        '''

        #positions = np.array(np.array([xdata, ydata, zdata]))
        FRAMES = np.shape(positions)[1]
        # Here positions has shape (10, 1089, 3)
        fig = plt.figure()
        ax = fig.add_subplot(111)
```

```
53
54          def init():
55                  ax.set_xlabel('step')
56                  ax.set_ylabel(chr(include + 120))
57
58          # animation function.  This is called sequentially
59          def animate(i):
60                  current_index = int(positions.shape[1] / FRAMES * i)
61                  ax.cla()
62                  ax.set_xlabel('step')
63                  ax.set_ylabel(chr(include + 120))
64                  # For line plot uncomment the following line
65                  # ax.plot3D(positions[:current_index, 0], positions[:current_index, 1],
                    ↪  positions[:current_index, 2])
66                  for position in positions:
67                  ax.scatter(np.arange(len(position))[:current_index],
                    ↪  position[:current_index, include])
68
69                  # call the animator.
70                  anim = animation.FuncAnimation(fig, animate, init_func=init,
                    ↪  frames=FRAMES, interval=100)
71
72                  return anim
```

Running the animation generation.

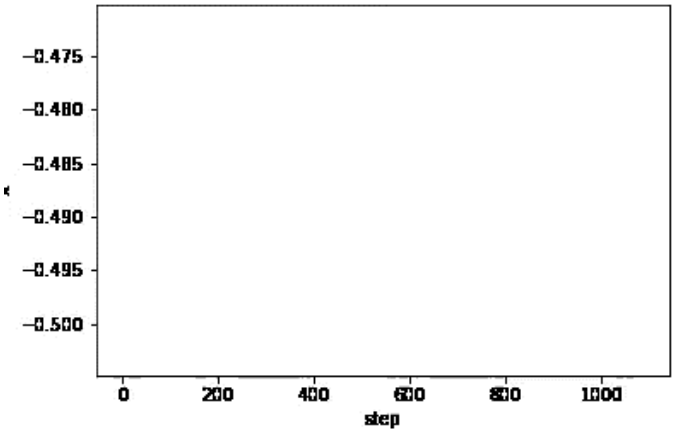
```
# Animate the positions
s1_allfield_10p_ps_anim = multiplot_animation_3d(s1_allfield_10p_ps)
display_animation(s1_allfield_10p_ps_anim)
s1_allfield_10p_ps_anim.save(r'multips1.mp4')

# x positions
s1_allfield_10p_ps_x_anim = multiplot_animation_1d(s1_allfield_10p_ps, include=0)
display_animation(s1_allfield_10p_ps_x_anim)
s1_allfield_10p_ps_x_anim.save(r'multipsx1.mp4')

# y positions
s1_allfield_10p_ps_y_anim = multiplot_animation_1d(s1_allfield_10p_ps, include=1)
display_animation(s1_allfield_10p_ps_y_anim)
s1_allfield_10p_ps_y_anim.save(r'multipsy1.mp4')

# z positions
s1_allfield_10p_ps_z_anim = multiplot_animation_1d(s1_allfield_10p_ps, include=2)
display_animation(s1_allfield_10p_ps_z_anim)
s1_allfield_10p_ps_z_anim.save(r'multipsz1.mp4')

# Animate the velocities
s1_allfield_10p_vs_anim = multiplot_animation_3d(s1_allfield_10p_vs)
display_animation(s1_allfield_10p_vs_anim)
s1_allfield_10p_vs_anim.save(r'multivs1.mp4')
```

Multiparticle position Animation

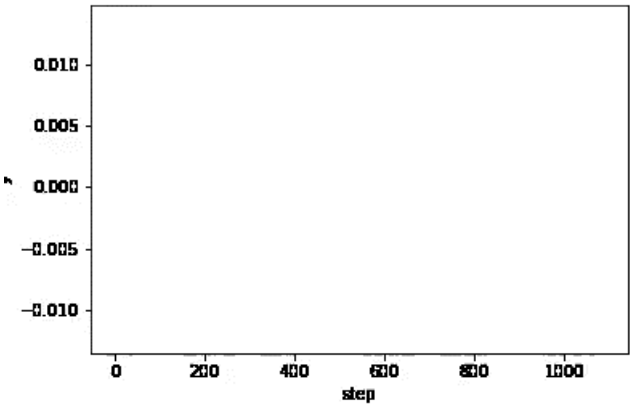


Multiparticle position components' Animation

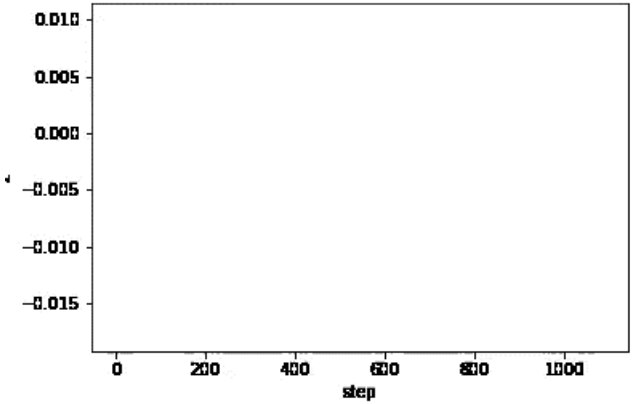
Multiparticle Position in x component



Multiparticle Position in y component



Multiparticle Position in z component



We can also look at the velocities.

```
# Plot the velocities
s1_allfield_10p_vs_fig = plt.figure()
s1_allfield_10p_vs_ax = plt.axes(projection='3d')
s1_allfield_10p_vs_ax.view_init(20, -120)

# Data for three-dimensional scattered points
for i in range(len(s1_particles)):
        s1_allfield_ap_vs_zdata = [elem[2] for elem in s1_allfield_10p_vs[i]] # Animate
        ↪  this plot as well.
        s1_allfield_ap_vs_xdata = [elem[0] for elem in s1_allfield_10p_vs[i]]
        s1_allfield_ap_vs_ydata = [elem[1] for elem in s1_allfield_10p_vs[i]]
        s1_allfield_10p_vs_ax.scatter3D(s1_allfield_ap_vs_xdata,
        ↪  s1_allfield_ap_vs_ydata, s1_allfield_ap_vs_zdata,\
        c=s1_allfield_p0_vs_zdata);
plt.savefig('multivs1', dpi='figure', format='png')

# Plot x velocities
s1_allfield_10p_vs_x_fig = plt.figure()
s1_allfield_10p_vs_x_ax = plt.axes()
for i in range(len(s1_particles)):
        s1_allfield_ap_vs_xdata = [elem[0] for elem in s1_allfield_10p_vs[i]]
        s1_allfield_10p_vs_x_ax.scatter(np.arange(len(s1_allfield_ap_vs_xdata)),
        ↪  s1_allfield_ap_vs_xdata);
plt.savefig('multivsx1', dpi='figure', format='png')
```

```
22
23  # Plot y velocities
24  s1_allfield_10p_vs_y_fig = plt.figure()
25  s1_allfield_10p_vs_y_ax = plt.axes()
26  for i in range(len(s1_particles)):
27          s1_allfield_ap_vs_ydata = [elem[1] for elem in s1_allfield_10p_vs[i]]
28          s1_allfield_10p_vs_y_ax.scatter(np.arange(len(s1_allfield_ap_vs_ydata)),
            ↪  s1_allfield_ap_vs_ydata);
29  plt.savefig('multivsy1', dpi='figure', format='png')
30
31  # Plot z velocities
32  s1_allfield_10p_vs_z_fig = plt.figure()
33  s1_allfield_10p_vs_z_ax = plt.axes()
34  for i in range(len(s1_particles)):
35          s1_allfield_ap_vs_zdata = [elem[2] for elem in s1_allfield_10p_vs[i]]
36          s1_allfield_10p_vs_z_ax.scatter(np.arange(len(s1_allfield_ap_vs_zdata)),
            ↪  s1_allfield_ap_vs_zdata);
37  plt.savefig('multivsz1', dpi='figure', format='png')
```

We get the following plots for the velocities

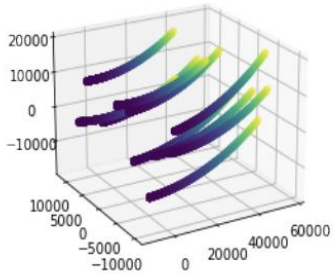


Figure 15: velocities

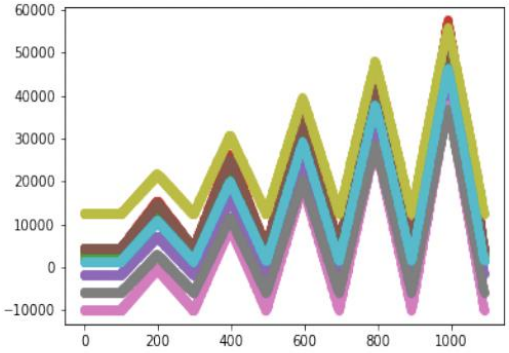


Figure 16: x-component of velocities

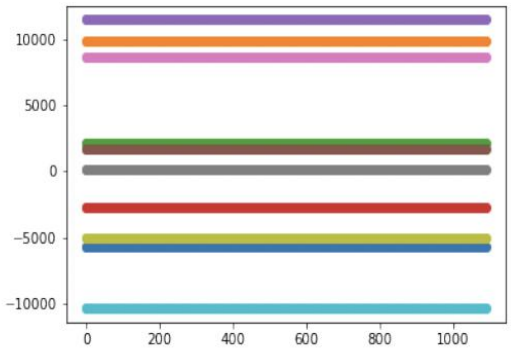


Figure 17: y-component of velocities

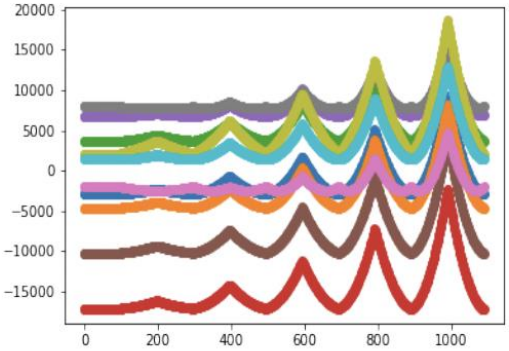
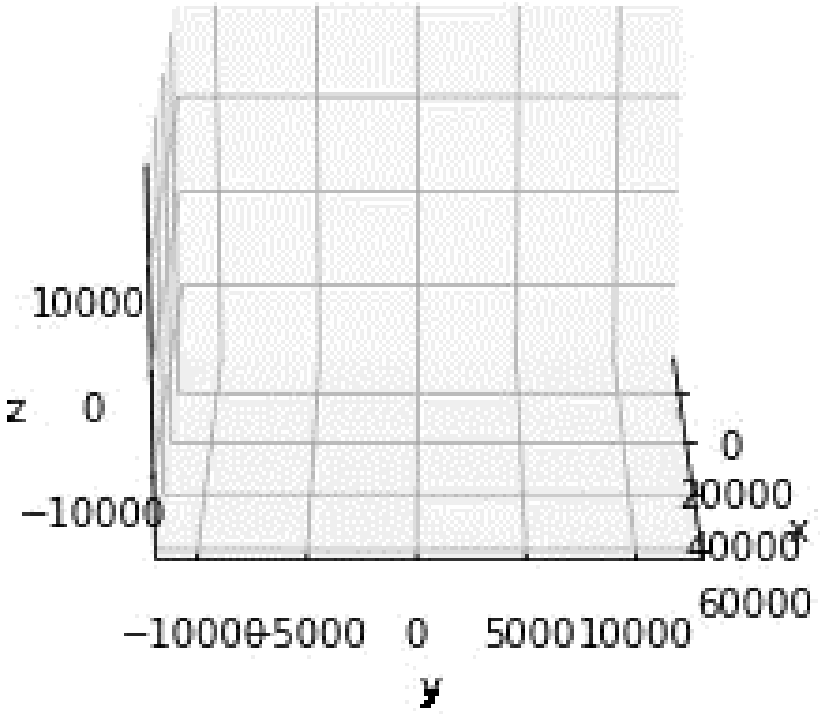


Figure 18: z-component of velocities

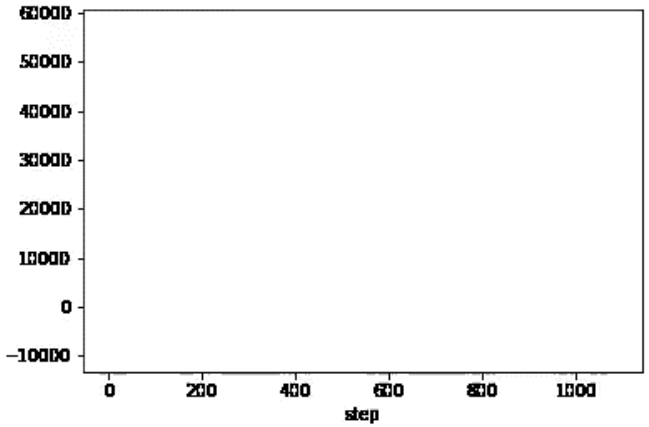
```
20
21  # Animate the velocities
22  s1_allfield_10p_vs_anim = multiplot_animation_3d(s1_allfield_10p_vs)
23  display_animation(s1_allfield_10p_vs_anim)
24  s1_allfield_10p_vs_anim.save(r'multivs1.mp4')
25
26  # x velocities
27  s1_allfield_10p_vs_x_anim = multiplot_animation_1d(s1_allfield_10p_vs, include=0)
28  display_animation(s1_allfield_10p_vs_x_anim)
29  s1_allfield_10p_vs_x_anim.save(r'multivsx1.mp4')
30
31  #y velocities
32  s1_allfield_10p_vs_y_anim = multiplot_animation_1d(s1_allfield_10p_vs, include=1)
33  display_animation(s1_allfield_10p_vs_y_anim)
34  s1_allfield_10p_vs_y_anim.save(r'multivsy1.mp4')
35
36  # z velocities
37  s1_allfield_10p_vs_z_anim = multiplot_animation_1d(s1_allfield_10p_vs, include=2)
38  display_animation(s1_allfield_10p_vs_z_anim)
39  s1_allfield_10p_vs_z_anim.save(r'multivsz1.mp4')
```

Multiparticle Velocity Animation

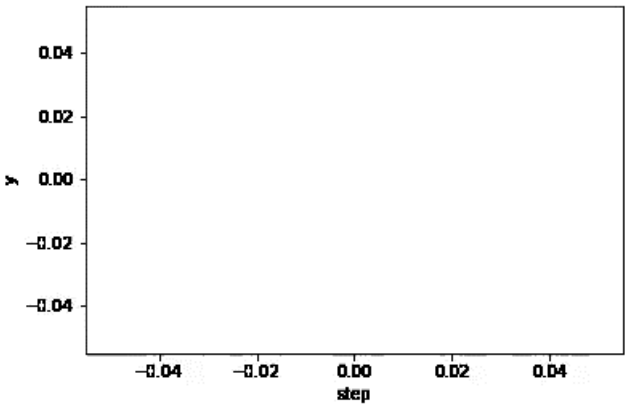


Multi-particle Velocity Components Animation

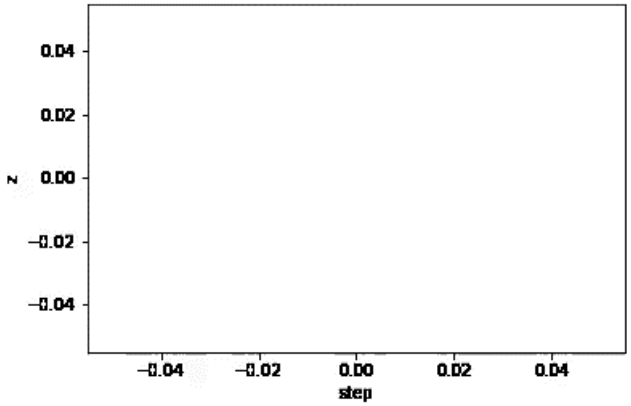
Multiparticle Velocity in x component



Multiparticle Velocity in y component



Multiparticle Velocity in z component



References

[1] Florian LB (GitHub user). Charged Particle Trajectories in Electric and Magnetic Fields. Thu, 28 Jan 2016. https://flothesof.github.io/charged-particle-trajectories-E-and-B-fields.html