**Magnetic Mirror Effect in Magnetron Plasma:  
Modeling of Plasma Parameters**

*Submitted in partial fulfillment of the requirements for the degree of*

Bachelor of Technology

in

**Mechanical Engineering**

*by*

## 18BEM0145 Sashi Kant Shah 18BME2104 Kaushal Timilsina 18BME2109 Hrishav Mishra

**Under the guidance of  
 Prof. / Dr. Sitaram Dash**

**School of Mechanical Engineering**   
 VIT, Vellore.

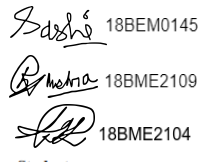


February 15,2022

**DECLARATION**

We hereby declare that the thesis entitled “Magnetic Mirror Effect in Magnetron Plasma: Modeling of Plasma Parameters" submitted by us, for the award of the degree of *Bachelor of Technology in Mechanical Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Professor Sitaram Dash.

We further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.



Place : Vellore   
Date : 15th February 2022  **Signature of the candidate**

**CERTIFICATE**

This is to certify that the thesis entitled “Magnetic Mirror Effect in Magnetron Plasma: Modeling of Plasma Parameters” submitted by **18BEM0145-Sashi Kant Shah, 18BME2104 Kaushal Timilsina &18BME2109 Hrishav Mishra**, VIT University, for the award of the degree of *Bachelor of Technology in Mechanical Engineering*, is a record of bonafide work carried out by him under my supervision during the period, 01. 12. 2021 to 30.04.2022, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 15th February 2022 **Signature of the Guide**

#### Internal Examiner External Examiner

Head of the Department Mechanical Engineering

## ACKNOWLEDGEMENTS

We would like to express our special gratitude to our project guide Prof. Sitaram Dash for his personal support and advice to carry out the thesis and complete it. We would further like to extend our gratitude to the School of Mechanical Engineering and VIT for providing us with this opportunity to enhance our previous semester knowledge practically through this capstone project and strengthen our understanding through the thesis for the same. We have got to learn a lot more about this project regarding the understanding and improvement of practical applications of Magnetron sputtering which will be very helpful for us as we progress in our field of study. We are also grateful for having the opportunity of learning team work which has helped us in recognizing our potential both on an individual as well as a team level. Being able to understand how each and every one of us are able to contribute through our individual thinking and building the outcome through amalgamation of the same is this something we are truly thankful for.

18BEM0145 Sashi Kant Shah   
18BME2104 Kaushal Timilsina   
18BME2109 Hrishav Mishra

# Executive Summary

The aim of this work is to understand the magnetic mirror configuration that is ideal in in a magnetic plasma trap chamber which can be used in Magnetron sputtering. Particles in a plasma have different speeds depending on the initial distribution which is based on parameters like the plasma temperature. The speeds of particles change depending on the electric and magnetic fields. Based on the magnetic mirror effect, one can determine which particle (having certain velocities) can escape the magnetic trap and which of those are reflected. The less the particles escape the magnetic trap, the more of the flux is used in forming coatings and less of the ionized gas is wasted. This is very useful in understanding the required gas supply and rate of deposition.

The model used in the project is mostly similar to the single particle model, where in a collection of particles, each particle evolves under the influence of the electric and magnetic fields set up by the apparatus; governed by the Lorentz force. However, we incorporate a little bit of the Kinetic theory in that we are interested in different initial velocity distributions for particles in the plasma and how the velocity distribution changes over time. To achieve this in a simulation, the Lorentz force equations are discretized and then solved using the Boris Algorithm; a standard algorithm for simulating charged particles in electric and magnetic fields.

**Key words:** Magnetic mirror, Magnetron sputtering **,** Lorentz force , Kinetic theory, Boris Algorithm

**CONTENTS Page**

**No.**

#### Acknowledgement i

[Executive Summary ii](#_TOC_250004)

Table of Contents Iii

[List of Figures ix](#_TOC_250003)

[List of Tables xiv](#_TOC_250002)

Abbreviations xvi

[Symbols and Notations xix](#_TOC_250001)

1. [INTRODUCTION 1](#_TOC_250000)
   1. Objective 1
   2. Motivation 1
   3. Background 1
2. PROJECT DESCRIPTION AND GOALS 1
3. TECHNICAL SPECIFICATION 2
4. DESIGN APPROACH AND DETAILS (as applicable) 4
   1. Design Approach / Materials & Methods 5
   2. Codes and Standards 6
   3. Constraints, Alternatives and Tradeoffs 8

#### SCHEDULE, TASKS AND MILESTONES 9

1. **PROJECT DEMONSTRATION 11**
2. RESULT & DISCUSSION 13

|  |
| --- |
| 1. **SUMMARY 40** |
| **9 REFERENCES 45** |
| **APPENDIX A**  **List of Figures**   |  |  |  |  | | --- | --- | --- | --- | | **Figure No.** |  | **Title** | **Page No.** | | 1 |  | Magnetron Sputtering | 41 | | 2  3  4  5  6  7  8  9  10  11  12  13  14 |  | Simulation Flow Chart  Position plot  X Position  Y Position  Z position  Velocity  X Velocity  Y Velocity  Z Velocity  X position  Y position  Z position | 41  42  42  42  42  43  43  43  43  44  44 |  List of Tables **Table No. Title Page No.**   |  |  |  |  | | --- | --- | --- | --- | | 1 | Gantt Chart |  | 44 | |

1. **INTRODUCTION**

**OBJECTIVE**

The objectives of the project define the structure of our algorithm and are as follows:

**1. Single Particle Method**   
To simulate charged particles that evolve under the influence of electric and magnetic fields; as governed by the Lorentz force.

**2. Field Configurations**   
To simulate a few different configurations of electric and magnetic fields- some describing apparatus like coils; some describing analytic expressions for fields and to study the different evolution of particles.  
 **3. Kinetic Theory**  
 To study different initial velocity distributions and how the velocity distribution of particles changes as the particles dynamically evolve. Parameters like the Plasma temperature are to be studied under this topic.  
**4. Analysis**   
To analyze different batches or collections of particles, subjected to different field configurations.

**1.2 MOTIVATION**  
Currently large number of researches going on to use of plasma in daily practical life, and as the theme of our project lies on using magnetic field and electric field to study the efficiency of magnetic sputtering, it gave us the motivation to choose the topic for our thesis. Magnetron sputtering now makes a significant impact in application areas including synthesis of hard, wear-resistant coatings, low friction coatings, corrosion-resistant coatings, decorative coatings and coatings with specific optical, or electrical properties.  DC magnetron sputtering can be used for synthesizing Ag nanoparticles in which their size can be precisely controlled.

**1.3 BACKGROUND**Our Institute has a lot of facilities for learning and researching upon Engineering Physics which is one of the core subjects in Mechanical Engineering. The physical principles governing practical applications are significant to enhance design capabilities and augment utilization aspects. This provides the starting push to choose Magnetic Mirror Effect as our project, apart from which the enthusiasm we shared in learning and understanding plasma physics and engineering. This is the basis of selection of our topic of research for the Capstone Project.

**2.PROJECT DESCRIPTION AND GOALS**

**a. Description**

A magnetic mirror configuration is important in a magnetic plasma trap chamber such as that used in Magnetron sputtering. Particles in a plasma have different speeds depending on the initial distribution which is based on parameters like the plasma temperature. The speeds of particles change depending on the magnitude of applied electric and magnetic fields. Based on the magnetic mirror effect, one can determine which particle (having certain velocities) can escape the magnetic trap and which of those are reflected. The less the particles escape the magnetic trap, the more of the flux is used in forming coatings and less of the ionized gas is wasted. This is very useful in understanding the required gas supply and rate of deposition.

The kinetic energy and the magnetic moment of a charged particle in a magnetic field are conserved if there are no external sources. If the magnetic field has a gradient, the   
component of the velocity of the particle parallel to the magnetic field changes and might

decrease to zero and increase in the opposite direction: which is observed as the particle

being reflected. This setup is important to understand and control how the ions are contained in and lost from a magnetron sputtering chamber.

**1**

**b. Goals**

•To simulate charged particles that develop affected by electric and attractive fields; as

administered by the Lorentz power.

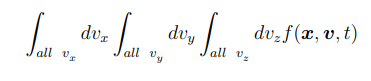
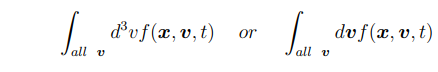
•To simulate a couple of designs of electric and attractive fields-some portraying devices like curls; some depicting logical articulations for fields and to concentrate on the different development of particles.

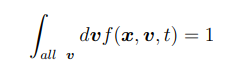
•To concentrate on various introductory speed circulations and how the speed dispersion of particles changes as the particles advance. Boundaries like the Plasma temperature are to be considered under this subject.

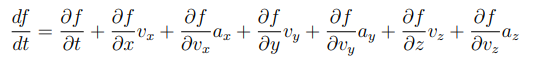
•To investigate various groups or assortments of particles, exposed to various field configurations.

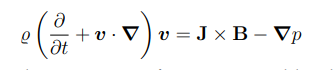
**3.TECHNICAL SPECIFICATION**

**1 Plasma as a system**   
Various models are used to describe Plasma as a system. Some of the common approaches are:

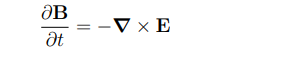
1. **Single particle description**  
 This model is used to describe the motion of a charged particle under the influence of electric and magnetic fields. The particle’s motion is described by the Lorentz force   
   
Describing many particles evolving under the influence of Lorentz force, it is easy to describe a plasma assuming that the mean free path of the particles is much larger than the dimensions of the chamber; meaning that particles hardly ever collide. This is to say that in the simplest case, under this model particle evolve under the influence of electric and magnetic fields but the particles do not produce any electric and magnetic fields of their own or affect the external applied fields and hence also do not interact with other particles.  
**2. Kinetic theory**   
The kinetic theory describes the plasma as collection of particles whose state (position and velocity) is treated as a random variable with a density function *f* (*x, y, z, vx, vy, vz, t*) which describes the number of particles at position (x, y, z) at time t with velocities between *vx* and *vx* + *dvx*, *vy* and *vy* + *dvy*, *vz* and *vz* + *dvz* in directions x, y and z respectively. For a simpler notation *f* (*x, y, z, vx, vy, vz, t)* is denoted as *f* (***x****,* ***v****, t*). The expression  
  
  
gives the number of particles at position x, at time t. For a simple choice of notation, it if often written as  
  
A density function is said to be normalized if

   
Such a density function is also denoted with a hat as *f*ˆ(***x****,* ***v****, t*).  
 **2**

The average velocity *v*¯ for a density function *f*ˆ(***x****,* ***v****, t*) is calculated as  
  
Various other features of the distribution are: the Root Mean Square velocity vrms, the average absolute velocity |v| , the average velocity in z direction |vz|, etc. The evolution of the particles is described as the changing of the density function. Boltzmann equation is often used to describe this dynamics:  
  
While it may look complicated at first, the left hand side of the equation is actually just a short form for:  
  
For simplicity, let us look at the 1-dimensional (2-phase dimensional) version of this expression  
  
If the right hand side of the Boltzmann equation were zero, in 1-dimension (2-phase dimensions), it would read:  
  
This would mean that the distribution function is unchanging (in its own frame). If particles were not interacting and there was no external disturbance, this is exactly what would happen. This is why with the expression on the right hand side set to zero, the Boltzmann equation is said to be collision less. Adding a non-zero expression on the right hand side would account for interactions between the particles or the effect of external disturbances.  
  
**3.Fluid model**   
The fluid model is describes the plasma in terms of macroscopic variables like Pressure, temperature, average velocity, density, flux; like ordinary fluid dynamics. The governing Partial Differential Equations are obtained by taking moments of the Boltzmann equation, or a special case of the Boltzmann equation called the Vlasov equation. Such a procedure gives rise to:   
The continuity equation:  


and the Cauchy momentum equation:  
  
Higher moments yield other equations for quantities like the entropy. The fluid equations average over the velocity distribution of the particles to obtain the macroscopic variables like Pressure and Temperature, as discussed earlier- by taking moments of the Kinetic equations like the Boltzmann equation or the Vlasov equation.  
 **3**

**4. Magnetohydrodynamics**

Magneto hydrodynamics(MHD) is an extension of the fluid model of describing a plasma. In a first approximation, the magnetic and electric fields acting on the fluid and the currents generated are now solved using Faraday’s law:  
  
and the Ampere’s law:  
  
The full set of Maxwell’s equations can also be coupled with the fluid. MHD description of a plasma also describes dynamic waves such as Alfvén waves and magneto sonic waves. MHD theory can also be used to describe interaction of the plasma with electromagnetic waves, for example those from a LASER source relevant to LASER plasma processes in surface engineering. Ideal MHD studies plasma as a fluid of single species, however, multi-species plasma can also be studied

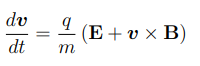
**5. The model used in the project**

In the context of plasma processes like Magnetron Sputtering where particles have a considerably large mean free path, we have decided not to use the fluid model and the MHD descriptions of a plasma. In such processes, it is almost always the case that the strength of the electric and magnetic fields applied by the apparatus is far stronger than the electric and magnetic fields generated by charged particles in a plasma. This approximation allows us to avoid solving Maxwell’s equations and use only the Lorentz force. The model used in the project is mostly similar to the single particle model, where in a collection of particles, each particle evolves under the influence of the electric and magnetic fields set up by the apparatus; governed by the Lorentz force. However, we incorporate a little bit of the Kinetic theory in that we are interested in different initial velocity distributions for particles in the plasma and how the velocity distribution changes over time; as it is important to understand the velocity distribution to characterize the reflection and loss of particles in a magnetic-mirror like trap that may be setup in the plasma chamber.

**4. DESIGN APPROACH AND DETAILS   
4.1 Design Approach / Materials & Methods**  
As discussed earlier, we evolve individual particles in the plasma with the Lorentz force. To achieve this in a simulation, the Lorentz force equations are discretized and then solved using the Boris Algorithm; a standard algorithm for simulating charged particles in electric and magnetic fields.

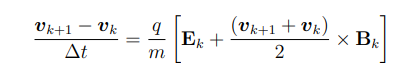
**Lorentz Force**

The equations of motion for a charged particle under the influence of Electric and Magnetic fields is described by the Lorentz Force in the S.I. units as



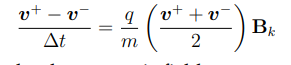
as discussed earlier, along with the expression for the velocity.

  
 **4**

These equations are discretized to obtain  
  
from the Lorentz Force equation, and  
  
from the expression for velocity, where the subscript k denotes the kth time step.

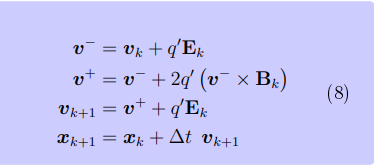
**Boris Algorithm**

The discretized Lorentz equations may be solved using the Boris Algorithm, as we do in this project. The Boris Algorithm splits the discretized Lorentz equation into three equations.

4  
  
which is often called the first half of the electric pulse.  
  
which is often called rotation by the magnetic field.  


which is almost the discretized Lorentz equation except that (***v***+ + ***v****−*) is substituted for (***v****k*+1 + ***v****k*). However, subtracting [the first electric pulse e](#_bookmark19)quation from equation gives (***v***+ + ***v****−*) = (***v****k*+1 + ***v****k*), giving the discretized Lorentz equation.This means that the Boris algorithm is equivalent to the discretized Lorentz equation.

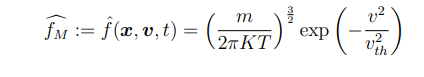
The equations of the Boris Algorithm can be written slightly different as

  
along with discretized position equation where  These equations are used to update the velocity of the particle under the influence of the Lorentz force under the Boris update strategy.

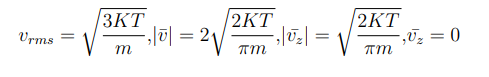
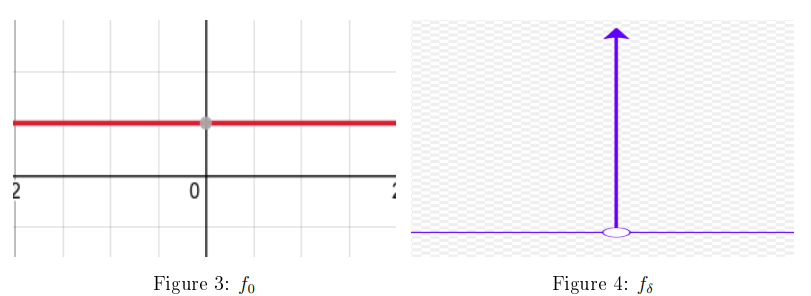
**2 Particle Sampling- kinetic theory**   
As discussed earlier, kinetic theory of plasma uses density function and its evolution to describe a plasma. We pointed out that we will be using some aspects of kinetic theory in our project; in that we will initialize the batches (collections) of particles based on certain distributions. This will allow us to study parameters like the plasma temperature, in our project  
 **5**

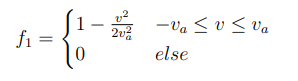
**Maxwellian distribution**

One important density function often used in the kinetic theory of plasma is the Maxwell-Boltzmann distribution often called the Maxwellian which has the density function

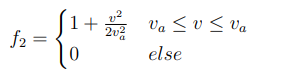
5  


Where  
   
Some features of the Maxwellian are:

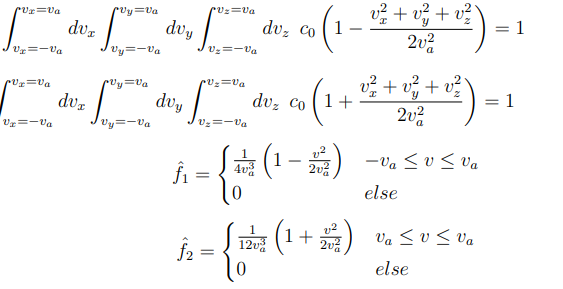
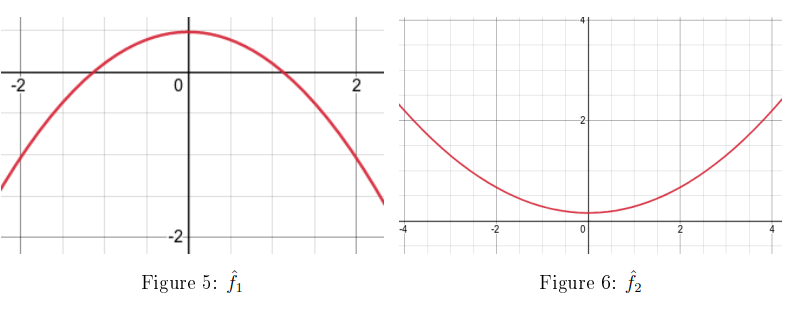
  
 **Parabolic distributions**  
 We are also interested in defining other density functions to do the sampling. The simplest density function to try would be to have all particles have the same velocity i.e. a Dirac delta function distribution. Another simple distribution would be a uniform distribution between two velocities, where a particle is equally likely to have any velocity in the range.  


Functions that look like f0 might define he uniform distribution while functions like fδ might describe the Dirac delta distribution. The figure for f0 was generated using desmos graphing calculator while the figure for fδ was snipped from the image in wikipedia for the Dirac delta function. We define initialization strategies based on uniform and Dirac delta distributions in our simulations.  
Parabolic functions in a given range also seem like interesting distributions and yet simple to work with. After some trial and error, two functions that seem interesting are:  
 

**6**



These functions can be normalized as

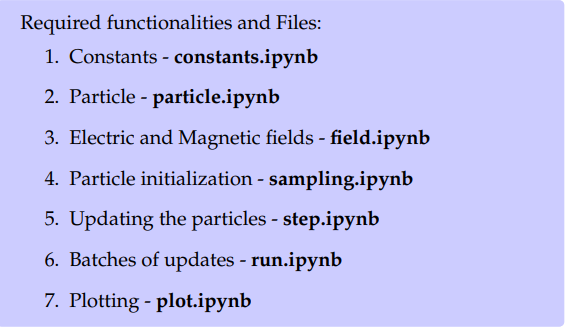
  
The graph of these functions (considering v as a single variable) were generated using desmos graphing calculator. The figures represent ˆf1 and ˆf2 extrapolated to beyond the defined range  
  
In the upper half of the plane, ˆf1 looks approximately like a Gaussian distribution which the Maxwellian is for a distribution of velocity (not speed) for a fixed temperature. ˆf2 is sort of the opposite of ˆf1. But given that the functions are defined in a limited range, the integrals do not diverge.  
  
**4.2 Codes and Standards**

The simulation is programmed in jupyter notebook files (.ipynb extensions) that run python and markdown. The notebooks are available in the following Github repository, in the notebooks folder.

**7**

**Github Repository**  
<https://github.com/18BME2104/MagneticMirror>

**Algorithm Outline**   
Different aspects of the program are discussed based on the different notebooks (ipynb files) that describe them.

  
 **Constants - constants.ipynb**   
  
The **constants.ipynb** notebook describes some constants useful in the program. Some useful constants are *e* (electron charge) *me* (electron mass), charges and masses of ions in the plasma, a.m.u (atomic mass unit), NA (Avogadro’s number),  (permittivity of vacuum), µ0 (permeability of vacuum), K or kB (Boltzmann’s constant), etc.  
 **Particle - particle.ipynb**   
  
The **particle.ipynb** notebook describes the state of a particle; its position, velocity, mass, charge, name, and optionally acceleration (which is set to 0 as default if it is not required to track the acceleration of a particle) as of now.   
Currently the Boris algorithm as discussed earlier, is an update strategy defined to update the state of a particle. Other strategies could be defined in new functions in the class. However, Boris algorithm is good enough for us to get started.

**Electric and Magnetic fields - field.ipynb**Electric and Magnetic field configurations described in **field.ipynb**.   
Currently **Uniform Electric field** and the **Radial Electric field** (the field depends on the particle’s position) created by an electrode are available to set up electric fields. Uniform Magnetic field and Magnetic field created by a Helmholtz coil and that by two Helmholtz coils are available.

Other Electric and Magnetic fields can be defined as functions in this class. These fields could be based on modeling of apparatus used to create electric or magnetic fields such as coils, or based on analytic expressions.

**8**

**Particle initialization - sampling.ipynb**  
 The initial positions and velocities of the particles play an important role in the evolution of their state under the influence of electric and magnetic fields. The initial distribution of positions and velocities define where the particles start in the setup (or lab apparatus); for example where they are injected into a sputtering chamber through valves, and what velocities they start with; for example what potential they are accelerated through or what parameters were used for the pumps used to pump the particles in.  
 7

The tracking of initial distribution is also important in determining how the magnetic mirror effect is observed in the plasma. The Sampler class samples initial positions and velocities for a given number of particles based on some available schemes.   
Currently positions can be sampled such that all particles start at the **same position** (like for example if injected through a port**), at the same distance** but randomly distributed in

angular positions relative to some point(origin as of now). Velocities can be sampled such that particles have the **same speed**, **same velocity** or **Maxwellian distributed speeds** or **Maxwellian distributed velocities**.

The initial positions and velocities could be passed around in lists but to be able to reuse some generated samples, and avoid sampling all the time, it is convenient to write sampled positions and velocities to files (csv format seems convenient). This functionality is described as:

**Updating the particles - step.ipynb**  
 The Step class is concerned with **initializing particles**, **defining the fields**, and **updating the states of the particles** (positions and velocities) under the action of electric and magnetic fields

To use sampled positions and velocities that have been saved in csv files, some reading functionality is useful to load these positions and velocities to be used during initialization.

**4.3 Constraints, Alternatives and Tradeoffs** The major portion of the work to be done to achieve this includes working on the **run.ipynb** notebook. To be able to understand the simulation; to check if works correctly and to understand the plasma behavior, we also need to define functionalities to plot different parameters like particle positions and velocities. These functionalities will be defined in **plot.ipynb** notebook, which can be imported into **ryn.ipynb** and **step.ipynb** notebooks to plot these quantities when the simulation is running, or from saved data.

**Batches of updates - run.ipynb**   
The Run class is concerned with running the steps defined by the Step class in **step.ipynb**. This includes creating batches of particles, updating them under the influence of electric and magnetic fields and removing them if they move out of the region of interest or for example are absorbed during a coating process. New batches of particles can be created to model the flow of particles as in a plasma chamber setup. Functionality to change the fields; for example changing the Voltages in electrodes or Currents in the coils, are also defined. This better models the control of plasma supply and electric and magnetic field control apparatus as in a laboratory.

**Plotting - plot.ipynb**   
The plot class is used to define functionalities to draw particle positions, plot positions and velocities of particles, as the system evolves like a lab apparatus. These functionalities are imported into run.ipynb; and if required other notebooks, to be used when the simulation is running; or optionally from saved data. Work on this notebook is yet to be started.

**9  
Understanding of the plasma**   
The big question, is that once we have a simulation running, how do we do analysis on the

plasma so that the project helps improve our understanding of a plasma system? The sampling aspect of the project, will allow us to understand how different parameters of the plasma like the temperature; when it enters the chamber affect the behavior of the plasma.

We can also study for example, the velocity distribution of the particles as they evolve in the chamber. Studying different field configurations; even simple functionalities like being able to change the Voltage of an electrode to change the electric field- like one might with a real apparatus, will help us understand the behavior of plasma under different or changing fields. Such strategies will allow us to understand how to contain a plasma in chamber; and one such instance would be a magnetron sputtering chamber where one could study the magnetic mirror effect. The magnetic mirror effect in a magnetron sputtering chamber serves as one example of studies we could do on the plasma parameters, studying a plasma in simulation. So our title serves as an example for what we could do with such a project. We have learned through the course of doing the project that more could be achieved with such a system than what we initially planned to focus on.

**5. SCHEDULE, TASKS AND MILESTONES**

**A. Schedule and Tasks:  
1. Weeks 1-2 Study: plasma models, kinetic theory**   
We first studied basic plasma physics, and a bit more on the kinetic theory of plasma as that seemed relevant to our project.

**2. Weeks 2-3 Study: PIC, Boris Algorithm**

We then studied particle in cell methods and then the Boris Algorithm.

**3.Weeks 3-4 Algorithm: idea, constants**

With some idea of how to proceed with the project, we started working on the algorithm. We started by with work on the constants.ipynb notebook, defining the constants that might be needed for the project.

**4. Weeks 4-5 Algorithm: particle, field, step**

We then worked on the notebook particle.ipynb where we defined a particle and how to update it. In the field.ipynb file, we worked to define some basic field configurations. Then we worked on step.ipynb to define steps of evolutions of the particles in a field configuration. This is the basic work on single particle dynamics.

**5. Weeks 5-6 Algorithm: sampling, Review preparation** We then worked on some kinetic theory aspects of the project, defining some sampling strategies based on the Uniform velocity and Maxwellian distribution. We then started preparing for review 1

**10**

**Review 2**

1. Weeks 7-8 **Algorithm: run, plot**  
 We plan to work next on run.ipynb to define running of batches of particle evolution and on plot.ipynb to define plot functionalities.   
2. Weeks 8-9 **Algorithm: changes and fixes**  
We anticipate that there may be errors in the programs that we will need some time to debug.   
3. Weeks 9-10 **First plasma simulation**   
We then expect to have the first plasma simulation running.  
 4. Weeks 10-11 **First results and study**

Running some plasma simulations, we intend to study the first set of results. 5. Weeks 11-12 Weeks 11-12 **Review preparation**We plan to prepare for review 2 based on some preliminary results and understanding.  
Review 3

1. Weeks 13-14 **Definitions: fields, density functions**   
We then intend to make some new definitions like sampling based on the parabolic density functions, some new field configurations and possibly some new functionality handling particle states.  
 2. Weeks 14-15 **Fixes and study**

We expect to spend some time making final fixes and changes on the algorithm. We then turn to studying the results.   
3. Weeks 15-16 **Study and result analysis**   
We intend to spend some time studying the results and trying to understand the plasma model in the simulation.   
4. Weeks 16-18 **Report and review preparation**

We then expect to start preparing for review 3, and preparation of the report and other documents.

**B. Milestones:**

**Milestone 1 : Algorithm start** ***Week 3 - Achieved***   
Around week 3 we had some understanding of basic plasma physics and so started working on how to formulate the problem as an algorithm. Before review 1   
**Milestone 2 :Simulation running *Week 9 - Expected***We expect to have the simulation running around week 9 of the project. This will allow us to have some preliminary results and understanding of our model, by review 2.

**11**

**Milestone 3: Study and understanding *Week 15 - Expected***We intend to perform study on plasma using our algorithm and improve our understanding of a plasma and how to control and study a plasma system before the end of the project. Before review 3

**6. PROJECT DEMONSTRATION**

**a. Particle Evolution**We evolve individual particles in the plasma with the Lorentz force. To achieve this in a simulation, the Lorentz force equations are discretized and then solved using the Boris Algorithm; a standard algorithm for simulating charged particles in electric and magnetic

fields. The equations of motion for a charged particle under the influence of Electric and Magnetic fields is described by the Lorentz Force in the S.I. units. The discretized Lorentz equations may be solved using the Boris Algorithm, as we do in this project

**b. Particle Sampling- kinetic theory**Kinetic theory of plasma uses density function and its evolution to describe a plasma. We pointed out that we will be using some aspects of kinetic theory in our project; in that we will initialize the batches (collections) of particles based on certain distributions. This will allow us to study parameters like the plasma temperature, in our project. One important density function often used in the kinetic theory of plasma is the Maxwell-Boltzmann distribution often called the Maxwellian. We are also interested in defining other density functions to do the sampling. The simplest density function to try would be to have all particles have the same velocity i.e. a Dirac delta function distribution. Another simple distribution would be a uniform distribution between two velocities, where a particle is equally likely to have any velocity in the range.

**c. Fields** In order to define the plasma chamber and hence the control on the plasma, we are interested in defining a few different electric and magnetic field configurations. We have a   
few different field configurations in mind. For the electric field, as of now we have 2 field configurations:

**1. Uniform electric field**   
The electric field has the same value everywhere. It is a very simple configuration to think about and use.

**2. Electric field due to an electrode** We describe the electrode in the following way: Every particle sees the electrode create an electric field created by a pair of capacitor plate at a distance taken only in the x-y plane. This is to say that every point on the electrode; now assumed to be a line along the z-direction, creates an electric field like that created between a

**12**

pair of capacitor plates, if there is a particle at the same z-coordinate as that point. So currently we use the expression E = V ·dr where dr is a vector whose z-coordinate is zero

For the magnetic field, we have 3 configurations as of now:

**1.Uniform magnetic field**The uniform magnetic field has the same value everywhere. It is very simple to understand and use.

**2. Magnetic field due to a Helmholtz coil**

The magnetic field strength due to a Helmholtz coil. We may use the axis of the coil as the direction of the field

**3. Magnetic field due to 2 Helmholtz coils**The magnetic field due to two Helmholtz coils is calculated by simply adding the magnetic fields created by the two coils. **d. Running different batches**

As discussed earlier we would like to describe different batches of particles, so as to model plasma streams; where the particles are sent into the chamber based on different initialization strategies. The particles are then evolved based on the Boris algorithm. We would then remove the particles to simulate particles exiting the chamber, or being absorbed to form coatings as in different surface engineering processes like Magnetron Sputtering.

**7. RESULT & DISCUSSION**

**7.1 Checks**

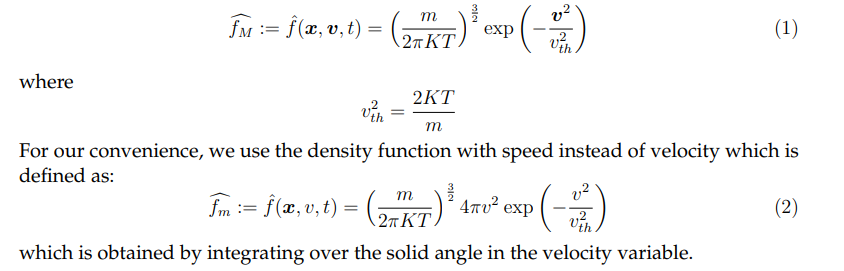
To verify that our program is working correctly, we perform some checks. We compare results from the program with calculations done by hand; of certain quantities and see if they agree. As of now, we are doing checks on the following aspects of the program:

1. Sampling
2. Update

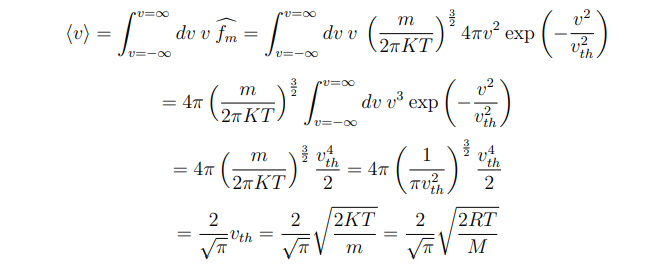
**7.1.1 Sampling**

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 agree with the calculations done by hand based on the plasma temperature. We recall the Maxwellian density function that we defined earlier.

**13**



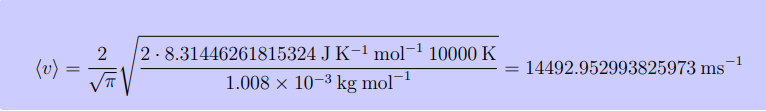
For this density function we calculate the average speed by with the expression:



We take the following data

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  
From section 1.1.1 Maxwellian sampling in the **checks.ipynb** notebook, we take the following:

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)

1

2

3

4

**14**

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

1

2

3

4

5

6

7

8

9

10

11

12

13

14

We get an average speed of the dis

From the above code, we get the following value for the average speed of sampled particles:

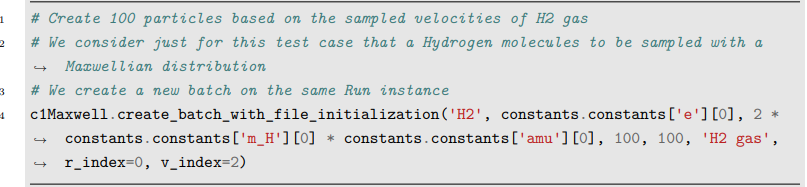
14202.764572898674 ms-1

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.

To be confident that our program indeed samples particles based on Maxwellian distribution as we expect it to, let’s check another distribution. For convenience, let’s assume Hydrogen molecules to be particles sampled with Maxwellian distribution, and check if the average speed of the sampled distribution agrees with the that expected from analytic calculation. For this we follow a similar procedure. We get the same temperature for this distribution but the relative molecular mass is doubled.

**15**

Initialization



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

1

2

3

4

5

6

7

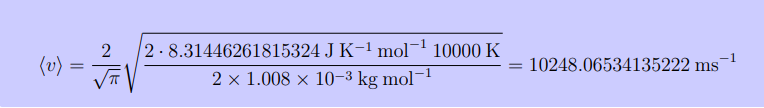
8

9

We get an average speed of the distribution to be:

10149.754879907316 ms-1

From analytic calculation, for the Hydrogen molecule particle distribution, we get



We see that the mean average speed of the sampled distribution and that expected

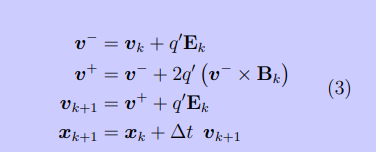
from the analytic expression are close for this distribution too; in fact, within 0*.*968% error. Because the errors are considerably small compared to statistical variation in the sampling, we believe the Maxwellian distribution sampling part of the program to be working as we expect it to.

## 16

## 7.1.2 Update

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

To verify that the simulation works as we expect it to, we perform a calculation and compare it to the output of an algorithm. From section 1.2.1 Boris Update in the **checks.ipynb** notebook, we take the following:



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)

1

2

3

4

Update input data

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

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*

1

2

3

4

5

6

7

**17**

Update

1

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]]

2

3

4

5

6

7

8

9

10

11

12

13

For particle at index 1, after the 3*rd* step of the update, the velocity of particle 1 was:

[39771.83801956555, -5029.491038, 4533.343932509505] ms-1

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

[48909.86581773698, -5029.491038, 8798.399243869582] ms-1

Its position after the 3*rd* step of the update was:

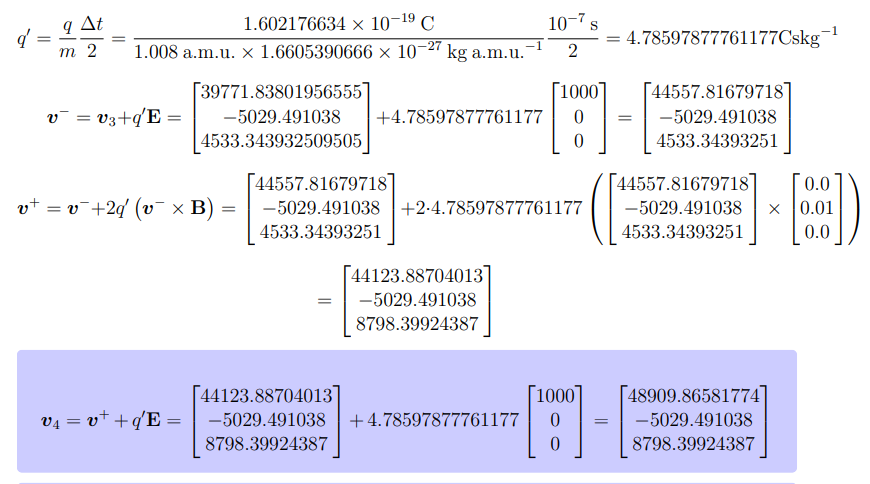
[-0.48985112694809785, -0.0020117964152, 0.00017005183797547688] m

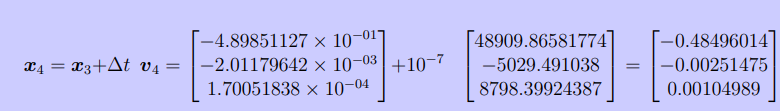
which was updated after the 4*th* step of the update to:

[-0.4849601403663242, -0.0025147455189999997, 0.001049891762362435] m

**18**

We now check if the same if true, with a calculation of the update done by hand.





We see that the values for ***x*3**, ***x*4**, ***v*3** and ***v*4** 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 3*rd* step of the update, the velocity of the particle was:

[38895.85871094631, -8670.854777, 13914.969423620274] ms-1

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

[47135.881299118584, -8670.854777, 18096.176367366777] ms-1

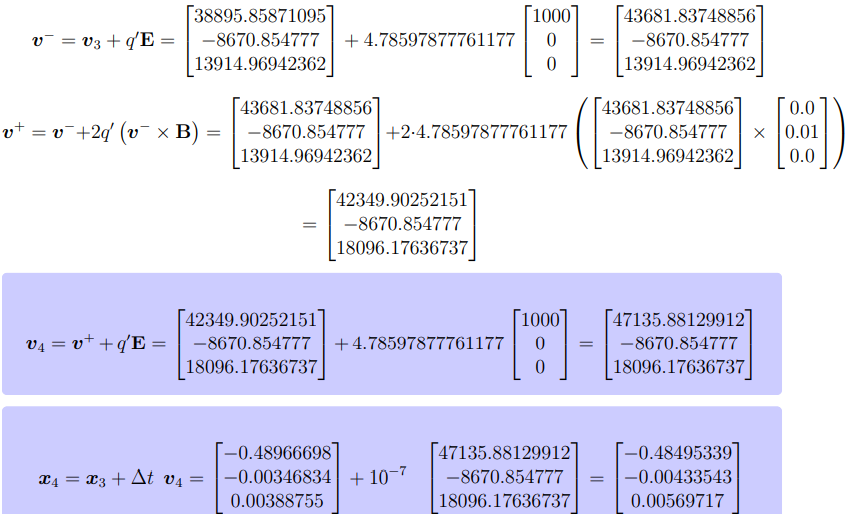
Its position after the 3*rd* step of the update was



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



**19**



We see that the values for ***x*3**, ***x*4**, ***v*3** and ***v*4** for particle 7 are close (the latter digitsdiffer due to the differing precision of calculations); i.e. the same when done by hand and in the program.

Based on these calculations, we believe that our particle update works as expected; according to the Boris Algorithm, and this part of the program works correctly.

**7.2 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 discuss the program written in **study1.ipynb**.

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)

After making the imports, we first create the system as in a batch of particle of a run object instance.

*#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)

1

2

3

4

5

6

7

8

**20**\

After creating a batch of particle, we take that batch and update it under the action of Electric and Magnetic fields. 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 and get the positions and velocities of the particles during the history of updates.

1 *# Take the first batch in this run object*

2 s1\_batch1 = s1.batches[0]['H ions']

3

4 *# Let’s consider Electric field being flipped in direction, and taken up a few scales*

5 E\_scale = [0, 0, 1, -1, 2, -2, 3, -3, 4, -4, 5, -5]

6 *# Let’s consider a case with constant Magnetic field*

7 B\_scale = [1 **for** i **in** range(len(E\_scale))]

8

9 s1\_index\_update = 0 *# Update the first batch in this Run instance*

10 s1\_particle\_track\_indices = [i **for** i **in** range(100)] *# Track all 100 particles*

11 s1\_dT = 10\*\*(-7) *# 0.1 microseconds*

12 s1\_stepT = 10\*\*(-9) *# 0.001 microseconds time step*

13 s1\_E0 = 1000 *# say 1000 Volts (voltage) per meter (size of chamber)*

14 s1\_Edirn = [1,0,0] *#in the x-direction [1,0,0]*

15 s1\_B0 = 10 \* (10\*\*(-3)) *# Meant to say 10 mT*

16 s1\_Bdirn = [0,1,0] *#in the y-direction [0,1,0]*

17

18 s1\_argsE = [element \* s1\_E0 **for** element **in** s1\_Edirn] *# currently the uniform\_E\_field*

*‹→ configuration is used*

19 s1\_argsB = [element \* s1\_B0 **for** element **in** s1\_Bdirn] *# currently the uniform\_B\_field*

*‹→ configuration is used*

20

21 s1\_batch\_ps\_and\_vs = dict()

22

23

**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

24

25

26

**21**

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.

1 s1\_descE\_is\_1 = 'E\_scale = ' + str(E\_scale[2]) + ' ' + 'B\_scale = ' + str(B\_scale[2])

2 s1\_descE\_is\_n1 = 'E\_scale = ' + str(E\_scale[3]) + ' ' + 'B\_scale = ' + str(B\_scale[3])

3 *# Extract update histories for two field configs*

4 s1\_histories\_E1 = s1\_batch\_ps\_and\_vs[s1\_descE\_is\_1]

5 s1\_histories\_nE1 = s1\_batch\_ps\_and\_vs[s1\_descE\_is\_n1]

6

7 *# Extract information on 0th particle’s update history in both cases*

8 s1\_descE\_is\_1\_p0 = s1\_histories\_E1[0]

9 s1\_descE\_is\_n1\_p0 = s1\_histories\_nE1[0]

10

11 *#Get positions and velocities of the particle’s update history*

12 s1\_descE\_is\_1\_p0\_ps = []

13 s1\_descE\_is\_1\_p0\_vs = []

14

15 **for** i **in** range(len(s1\_descE\_is\_1\_p0)):

16 s1\_descE\_is\_1\_p0\_ps.append(s1\_descE\_is\_1\_p0[i][1])

17 s1\_descE\_is\_1\_p0\_vs.append(s1\_descE\_is\_1\_p0[i][2])

18

19 s1\_descE\_is\_n1\_p0\_ps = []

20 s1\_descE\_is\_n1\_p0\_vs = []

21

22 **for** i **in** range(len(s1\_descE\_is\_n1\_p0)):

23 s1\_descE\_is\_n1\_p0\_ps.append(s1\_descE\_is\_n1\_p0[i][1])

24 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.

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()

2

3 s1\_descE\_is\_1\_p0\_ps\_ax = plt.axes(projection='3d')

4 s1\_descE\_is\_1\_p0\_ps\_ax.view\_init(50, -20)

5

6 *# Data for three-dimensional scattered points*

7 *# for position update history of particle 0 when the Electric field was scaled by 1*

8 s1\_descE\_is\_1\_p0\_ps\_zdata = [elem[2] **for** elem **in** s1\_descE\_is\_1\_p0\_ps]

9 s1\_descE\_is\_1\_p0\_ps\_xdata = [elem[0] **for** elem **in** s1\_descE\_is\_1\_p0\_ps]

10 s1\_descE\_is\_1\_p0\_ps\_ydata = [elem[1] **for** elem **in** s1\_descE\_is\_1\_p0\_ps]

11 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,\

12 c=s1\_descE\_is\_1\_p0\_ps\_zdata, cmap='Greens');

13 plt.savefig('EplusPs', dpi='figure', format='png')

14

15 *# Plot the position update history of particle 0 when the electric field is scaled by -1*

16 *# We see an opposite curve to the previous figure*

17 s1\_descE\_is\_n1\_p0\_ps\_fig = plt.figure()

18 s1\_descE\_is\_n1\_p0\_ps\_ax = plt.axes(projection='3d')

19 s1\_descE\_is\_n1\_p0\_ps\_ax.view\_init(50, -20)

20

21 *# Data for three-dimensional scattered points*

22 *# for position update history of particle 0 when the Electric field was scaled by -1*

23 s1\_descE\_is\_n1\_p0\_ps\_zdata = [elem[2] **for** elem **in** s1\_descE\_is\_n1\_p0\_ps]

24 s1\_descE\_is\_n1\_p0\_ps\_xdata = [elem[0] **for** elem **in** s1\_descE\_is\_n1\_p0\_ps]

25 s1\_descE\_is\_n1\_p0\_ps\_ydata = [elem[1] **for** elem **in** s1\_descE\_is\_n1\_p0\_ps]

26 s1\_descE\_is\_n1\_p0\_ps\_ax.scatter3D(s1\_descE\_is\_n1\_p0\_ps\_xdata,

*‹→* s1\_descE\_is\_n1\_p0\_ps\_ydata, \

27 s1\_descE\_is\_n1\_p0\_ps\_zdata, c=s1\_descE\_is\_n1\_p0\_ps\_zdata, cmap='Greens');

28 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.

**23**

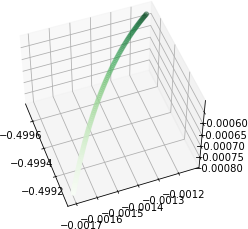
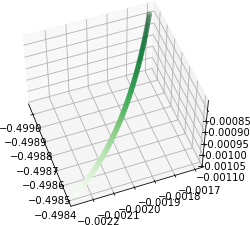
 

Figure 1: Electric field scaled by 1 Figure 2: Electric field scaled by -1

**Plotting particle update history**

One can notice that the particle seems to curve differently. To make things more clear, let’s plot the position of the particle during all of the used configurations of the electric field.

1

*# 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*

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

**24**

24

*# 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')

25

26

27

28

29

30

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

1

*# 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')

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

We get the following figures for the position and its components for the particle during the

**25**

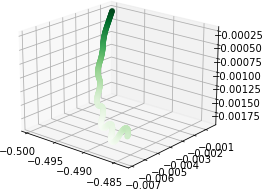
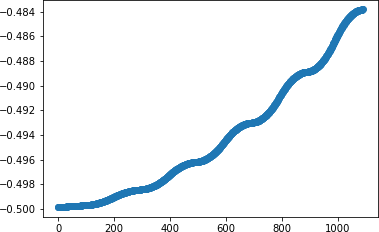
 

Figure 3: position Figure 4: *x*-component of position

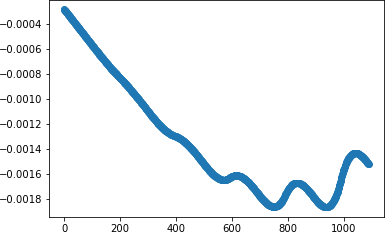
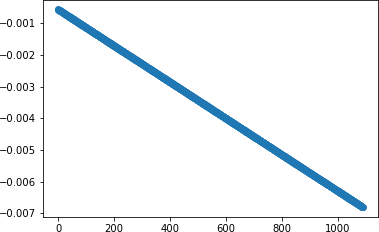


Figure 5: *y*-component of position Figure 6: *z*-component of position

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

1

*# 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]

2

3

4

5

6

7

8

9

10

**26**

11 s1\_allfield\_p0\_vs\_ax.scatter3D(s1\_allfield\_p0\_vs\_xdata, s1\_allfield\_p0\_vs\_ydata,

*‹→* s1\_allfield\_p0\_vs\_zdata,\

12 c=s1\_allfield\_p0\_vs\_zdata, cmap='Greens');

13 plt.savefig('vs1', dpi='figure', format='png')

14

15 *# Plot x velocity*

16 s1\_allfield\_p0\_vs\_x\_fig = plt.figure()

17 s1\_allfield\_p0\_vs\_x\_ax = plt.axes()

18 s1\_allfield\_p0\_vs\_x\_ax.scatter(np.arange(len(s1\_allfield\_p0\_vs\_xdata)),

*‹→* s1\_allfield\_p0\_vs\_xdata);

19 plt.savefig('vsx1', dpi='figure', format='png')

20

21 *# Plot y velocity*

22 s1\_allfield\_p0\_vs\_y\_fig = plt.figure()

23 s1\_allfield\_p0\_vs\_y\_ax = plt.axes()

24 s1\_allfield\_p0\_vs\_y\_ax.scatter(np.arange(len(s1\_allfield\_p0\_vs\_ydata)),

*‹→* s1\_allfield\_p0\_vs\_ydata);

25 plt.savefig('vsy1', dpi='figure', format='png')

26

27 *# Plot z velocity*

28 s1\_allfield\_p0\_vs\_z\_fig = plt.figure()

29 s1\_allfield\_p0\_vs\_z\_ax = plt.axes()

30 s1\_allfield\_p0\_vs\_z\_ax.scatter(np.arange(len(s1\_allfield\_p0\_vs\_zdata)),

*‹→* s1\_allfield\_p0\_vs\_zdata);

31 plt.savefig('vsz1', dpi='figure', format='png')

We get the following output.

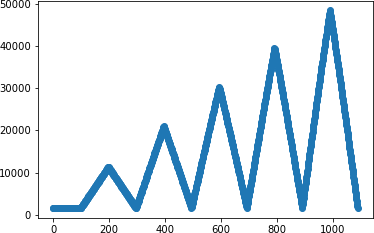
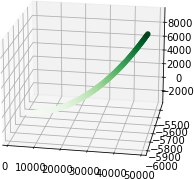


Figure 7: velocity Figure 8: *x*-component of velocity

**27**

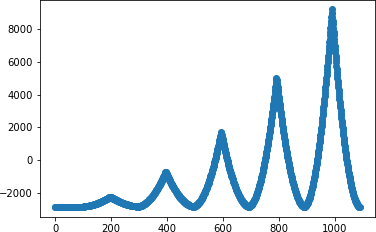
 

Figure 9: *y*-component of velocity Figure 10: *z*-component of velocity

**E** X **B Drift**

**E** x **B** drift is the drift of a particle in direction perpendicular to the electric and the magnetic fields, which is along the *z*-axis in this example. We can see the effect of **E** x **B** drift in figur[e(3)](#_bookmark0) and figur[e(6),](#_bookmark1) although we also see the effect of particle having a non- zero initial velocity in some direction. What we see in figur[e(3)](#_bookmark0) which is perhaps more clear in figur[e(6)](#_bookmark1) is that the particle is initially moving downwards along the *z*-axis. But following events of changing the sign of the electric field, the particle moves upwards until the next event of the change of sign of the electric field. As the strength of the electric field becomes greater later along the update, we see that the drifts cover longer distances.

We should be **careful** that figur[e(7)](#_bookmark2) hides the fact that the velocity is increasing and decreasing when we change the direction of the electric field, the points in the figure overlap and we do not see the distinction between multiple instances during the update when the same velocity is attained. Figures [(8)](#_bookmark2) and [(10)](#_bookmark3) help us understand this better. We also observe in figure [(9)](#_bookmark3) that the *y*-component of the velocity; along the magnetic field, stays constant.

A thing to notice is that we started by saying that we might want to consider a cube of size 1m x 1m x 1m; which would be a reasonable size for a plasma chamber, and we see that the particle stays well within the box. This means that such a time frame of plasma control might be relevant to processes in surface engineering where particles in hot plasma have some high velocity and participate in the processes in timescales of microseconds as in the example.

Animations have been generated for all plots and they help us understand these update histories better.

**28**

Animations were generated using the following parts of the program.

1

**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

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

**29**

34

**def** plot\_animation\_1d(positions, include):

*’’’*

*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 ’’’*

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)

**def** init():

ax.set\_xlabel('step') ax.set\_ylabel(chr(include + 120))

*# animation function. This is called sequentially*

**def** animate(i):

current\_index = int(positions.shape[0] / FRAMES \* i) ax.cla()

ax.set\_xlabel('step') ax.set\_ylabel(chr(include + 120))

*# For line plot uncomment the following line*

*# ax.plot3D(positions[:current\_index, 0], positions[:current\_index, 1],*

*‹→ positions[:current\_index, 2])*

ax.scatter(np.arange(len(positions))[:current\_index],

*‹→* positions[:current\_index, include])

*# call the animator.*

anim = animation.FuncAnimation(fig, animate, init\_func=init,

*‹→* frames=FRAMES, interval=100)

**return** anim

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

Running the animation generation.

**30**

Running the animation generation.

1

*# 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)

2

3

4

5

6

7

8 display\_animation(s1\_allfield\_p0\_ps\_x\_anim)

9 s1\_allfield\_p0\_ps\_x\_anim.save(r'psx1.mp4')

10

11 *# y positions*

12 s1\_allfield\_p0\_ps\_y\_anim = plot\_animation\_1d(s1\_allfield\_p0\_ps, include=1)

13 display\_animation(s1\_allfield\_p0\_ps\_y\_anim)

14 s1\_allfield\_p0\_ps\_y\_anim.save(r'psy1.mp4')

15

16 *# z positions*

17 s1\_allfield\_p0\_ps\_z\_anim = plot\_animation\_1d(s1\_allfield\_p0\_ps, include=2)

18 display\_animation(s1\_allfield\_p0\_ps\_z\_anim)

19 s1\_allfield\_p0\_ps\_z\_anim.save(r'psz1.mp4')

20

21 *# Plot the velocity*

22 s1\_allfield\_p0\_vs\_anim = plot\_animation\_3d(s1\_allfield\_p0\_vs)

23 display\_animation(s1\_allfield\_p0\_vs\_anim)

24 s1\_allfield\_p0\_vs\_anim.save(r'vs1.mp4')

25

26 *# Plot x velocity*

27 s1\_allfield\_p0\_vs\_x\_anim = plot\_animation\_1d(s1\_allfield\_p0\_vs, include=0)

28 display\_animation(s1\_allfield\_p0\_vs\_x\_anim)

29 s1\_allfield\_p0\_vs\_x\_anim.save(r'vsx1.mp4')

30

31 *# Plot y velocity*

32 s1\_allfield\_p0\_vs\_y\_anim = plot\_animation\_1d(s1\_allfield\_p0\_vs, include=1)

33 display\_animation(s1\_allfield\_p0\_vs\_y\_anim)

34 s1\_allfield\_p0\_vs\_y\_anim.save(r'vsy1.mp4')

35

36 *# Plot z velocity*

37 s1\_allfield\_p0\_vs\_z\_anim = plot\_animation\_1d(s1\_allfield\_p0\_vs, include=2)

38 display\_animation(s1\_allfield\_p0\_vs\_z\_anim)

39 s1\_allfield\_p0\_vs\_z\_anim.save(r'vsz1.mp4')

**31**

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):

In generating the animations for the plots, we took help from the blog post titled:

**Charged Particle Trajectories in Electric and Magnetic Fields** [([1]).](#_bookmark8)

1

2

3

4

5

6

**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))

7

8

9

10

11

12

13

14

15

16

17

18

19

## 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)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

24 s1\_allfield\_10p\_ps.append(s1\_allfield\_ap\_ps)

25 s1\_allfield\_10p\_vs.append(s1\_allfield\_ap\_vs)

26

27 s1\_allfield\_10p\_ps = np.array(s1\_allfield\_10p\_ps)

28 s1\_allfield\_10p\_vs = np.array(s1\_allfield\_10p\_vs)

29

30 *# Plot the positions*

31 s1\_allfield\_10p\_ps\_fig = plt.figure()

32 s1\_allfield\_10p\_ps\_ax = plt.axes(projection='3d')

33 s1\_allfield\_10p\_ps\_ax.view\_init(20, -80)

34

35 *# Data for three-dimensional scattered points*

36 **for** i **in** range(len(s1\_particles)):

37 s1\_allfield\_ap\_ps\_zdata = [elem[2] **for** elem **in** s1\_allfield\_10p\_ps[i]]

38 s1\_allfield\_ap\_ps\_xdata = [elem[0] **for** elem **in** s1\_allfield\_10p\_ps[i]]

39 s1\_allfield\_ap\_ps\_ydata = [elem[1] **for** elem **in** s1\_allfield\_10p\_ps[i]]

40 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);

41 plt.savefig('multips1', dpi='figure', format='png')

42

43 *# Plot x positions*

44 s1\_allfield\_10p\_ps\_x\_fig = plt.figure()

45 s1\_allfield\_10p\_ps\_x\_ax = plt.axes()

46 **for** i **in** range(len(s1\_particles)):

47 s1\_allfield\_ap\_ps\_xdata = [elem[0] **for** elem **in** s1\_allfield\_10p\_ps[i]]

48 s1\_allfield\_10p\_ps\_x\_ax.scatter(np.arange(len(s1\_allfield\_ap\_ps\_xdata)),

*‹→* s1\_allfield\_ap\_ps\_xdata);

49 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]].

**33**

64

s1\_allfield\_10p\_ps\_z\_ax.scatter(np.arange(len(s1\_allfield\_ap\_ps\_zdata)),

*‹→* s1\_allfield\_ap\_ps\_zdata); plt.savefig('multipsz1', dpi='figure', format='png')

65

We get the following plots of the positions:

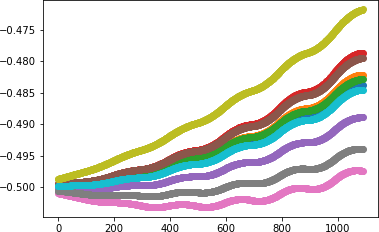
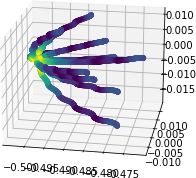


Figure 11: positions Figure 12: *x*-component of positions

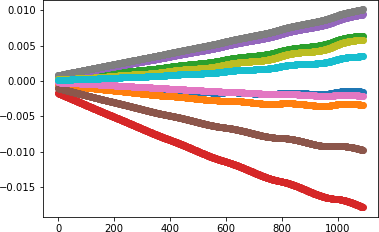
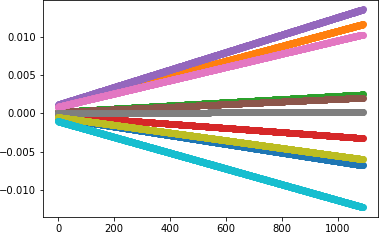


Figure 13: *y*-component of positions Figure 14: *z*-component of positions

In figure [(11),](#_bookmark4) we observe that all the particles begin at the same point and spread out based on their velocities. In figures [(12),](#_bookmark4) [(13)](#_bookmark5) and [(14),](#_bookmark5) we see that all the particles start at position [-0.5, 0, 0] as we intended them to when setting up the study; and spread out based on their initial velocities.

We can also look at the velocities.

**34**

1 *# Plot the velocities*

2 s1\_allfield\_10p\_vs\_fig = plt.figure()

3 s1\_allfield\_10p\_vs\_ax = plt.axes(projection='3d')

4 s1\_allfield\_10p\_vs\_ax.view\_init(20, -120)

5

6 *# Data for three-dimensional scattered points*

7 **for** i **in** range(len(s1\_particles)):

8 s1\_allfield\_ap\_vs\_zdata = [elem[2] **for** elem **in** s1\_allfield\_10p\_vs[i]] *# Animate*

*‹→ this plot as well.*

9 s1\_allfield\_ap\_vs\_xdata = [elem[0] **for** elem **in** s1\_allfield\_10p\_vs[i]]

10 s1\_allfield\_ap\_vs\_ydata = [elem[1] **for** elem **in** s1\_allfield\_10p\_vs[i]]

11 s1\_allfield\_10p\_vs\_ax.scatter3D(s1\_allfield\_ap\_vs\_xdata,

*‹→* s1\_allfield\_ap\_vs\_ydata, s1\_allfield\_ap\_vs\_zdata,\

12 c=s1\_allfield\_p0\_vs\_zdata);

13 plt.savefig('multivs1', dpi='figure', format='png')

14

15 *# Plot x velocities*

16 s1\_allfield\_10p\_vs\_x\_fig = plt.figure()

17 s1\_allfield\_10p\_vs\_x\_ax = plt.axes()

18 **for** i **in** range(len(s1\_particles)):

19 s1\_allfield\_ap\_vs\_xdata = [elem[0] **for** elem **in** s1\_allfield\_10p\_vs[i]]

20 s1\_allfield\_10p\_vs\_x\_ax.scatter(np.arange(len(s1\_allfield\_ap\_vs\_xdata)),

*‹→* s1\_allfield\_ap\_vs\_xdata);

21 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')

**35**

We get the following plots for the velocities:

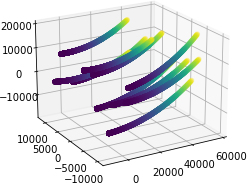
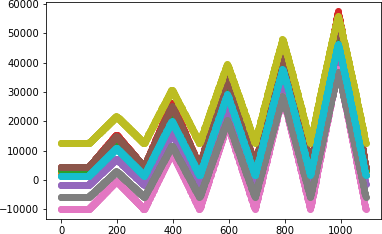
 

Figure 15: velocities Figure 16: *x*-component of velocities

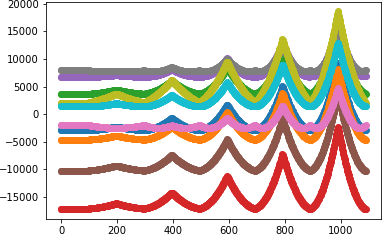


Figure 17: *y*-component of velocities Figure 18: *z*-component of velocities

We see that the particles start out with different values for each component and the *y*-component of the velocities of the particles stay the same. Like for a single particle, we also see the effect of changing the direction of the Electric field on the *x* and *z* components of the velocities. Also like the single particle plots, we see that the velocity oscillations observed in figures [(16)](#_bookmark6) and [(18)](#_bookmark7) are not observed in figure [(15)](#_bookmark6) because the points in the plot overlap. Animations of the plots might help us better understand the plots. Animations were generated using the following part of the program.

1

**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*

2

3

4

**36**

5

6 *This function can plot both positions and velocities*

7 *’’’*

8

9 *#positions = np.array(np.array([xdata, ydata, zdata]))*

10 FRAMES = np.shape(positions)[1]

11 *# Here positions has shape (10, 1089, 3)*

12 fig = plt.figure()

13 ax = fig.add\_subplot(111, projection='3d')

14

15 **def** init():

16 ax.view\_init(elev=20., azim=0)

17 ax.set\_xlabel('x')

18 ax.set\_ylabel('y')

19 ax.set\_zlabel('z')

20

21 *# animation function. This is called sequentially*

22 **def** animate(i):

23 current\_index = int(positions.shape[1] / FRAMES \* i)

24 ax.cla()

25 ax.view\_init(elev=20., azim=i)

26 ax.set\_xlabel('x')

27 ax.set\_ylabel('y')

28 ax.set\_zlabel('z')

29 *# For line plot uncomment the following line*

30 *# ax.plot3D(positions[:current\_index, 0], positions[:current\_index, 1],*

*‹→ positions[:current\_index, 2])*

31 **for** position **in** positions:

32 ax.scatter3D(position[:current\_index, 0], position[:current\_index, 1],

*‹→* position[:current\_index, 2])

33

34 *# call the animator.*

35 anim = animation.FuncAnimation(fig, animate, init\_func=init,

*‹→* frames=FRAMES, interval=100)

36

37 **return** anim

38

39 **def** multiplot\_animation\_1d(positions, include):

40 *’’’Here each element of positions is data for 1 particle that one would give as*

*‹→ input to*

41 *plot\_animation\_3d function, i.e. position or velocity update history of 1*

*‹→ particle*

42

43 *This function can plot both positions and velocities*

**37**

|  |  |  |
| --- | --- | --- |
| 44 | *include can be 0, 1 or 2.* |  |
| 45 | *if include = 2, this means plot the z data of the array* |
| 46 | *’’’* |
| 47 |  |
| 48 | *#positions = np.array(np.array([xdata, ydata, zdata]))* |
| 49 | FRAMES = np.shape(positions)[1] |
| 50 | *# Here positions has shape (10, 1089, 3)* |
| 51 | fig = plt.figure() |
| 52 | 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  66  67  68  69  70  71  72 | *# ax.plot3D(positions[:current\_index, 0], positions[:current\_index, 1],*  *‹→ positions[:current\_index, 2])*  **for** position **in** positions: ax.scatter(np.arange(len(position))[:current\_index],  *‹→* position[:current\_index, include])  *# call the animator.*  anim = animation.FuncAnimation(fig, animate, init\_func=init,  *‹→* frames=FRAMES, interval=100)  **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')

1

2

3

4

5

6

7

8

9

**38**

10

11 *# y positions*

12 s1\_allfield\_10p\_ps\_y\_anim = multiplot\_animation\_1d(s1\_allfield\_10p\_ps, include=1)

13 display\_animation(s1\_allfield\_10p\_ps\_y\_anim)

14 s1\_allfield\_10p\_ps\_y\_anim.save(r'multipsy1.mp4')

15

16 *# z positions*

17 s1\_allfield\_10p\_ps\_z\_anim = multiplot\_animation\_1d(s1\_allfield\_10p\_ps, include=2)

18 display\_animation(s1\_allfield\_10p\_ps\_z\_anim)

19 s1\_allfield\_10p\_ps\_z\_anim.save(r'multipsz1.mp4')

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')

In this first simple study, we were able to look at the behavior of a system of particles being updated under the influence of non-uniform electric field. In particular, we were able to observe the **E** x **B** drift.

**39**

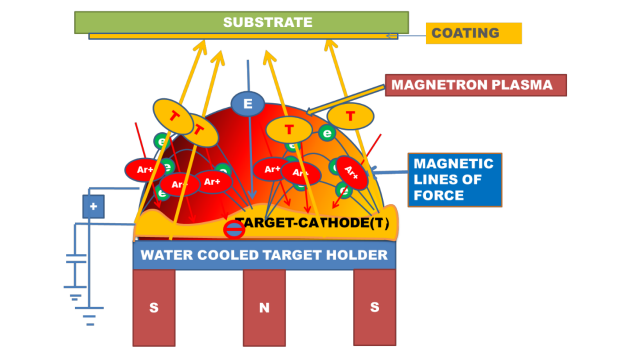
**8. SUMMARY**

The magnetic mirror effect can first be illustrated with a single particle. A magnetic mirror configuration is important in a magnetic plasma trap chamber such as that used in Magnetron sputtering. Particles in a plasma have different speeds depending on the initial distribution which is based on parameters like the plasma temperature. The speeds of particles change depending on the electric and magnetic fields. Based on the magnetic mirror effect, one can determine which particle (having certain velocities) can escape the magnetic trap and which of those are reflected. The less the particles escape the magnetic trap, the more of the flux is used in forming coatings and less of the ionized gas is wasted. This is very useful in understanding the required gas supply and rate of deposition

The model used in the project is mostly similar to the single particle model, where in a collection of particles, each particle evolves under the influence of the electric and magnetic fields set up by the apparatus; governed by the Lorentz force. However, we incorporate a little bit of the Kinetic theory in that we are interested in different initial velocity distributions for particles in the plasma and how the velocity distribution changes over time. To achieve this in a simulation, the Lorentz force equations are discretized and then solved using the Boris Algorithm; a standard algorithm for simulating charged particles in electric and magnetic fields.  
  
The sampling aspect of the project, will allow us to understand how different parameters of the plasma like the temperature; when it enters the chamber affect the behavior of the plasma. We can also study for example, the velocity distribution of the particles as they evolve in the chamber. Studying different field configurations; even simple functionalities like being able to change the Voltage of an electrode to change the electric field- like one might with a real apparatus, will help us understand the behavior of plasma under different or changing fields. Such strategies will allow us to understand how to contain a plasma in chamber; and one such instance would be a magnetron sputtering chamber where one could study the magnetic mirror effect. The magnetic mirror effect in a magnetron sputtering chamber serves as one example of studies we could do on the plasma parameters, studying a plasma in simulation. So our title serves as an example for what we could do with such a project. We have learned through the course of doing the project that more could be achieved with such a system than what we initially planned to focus on

**40**

**List of Figures**

Fig 1: Magnetron Sputtering System

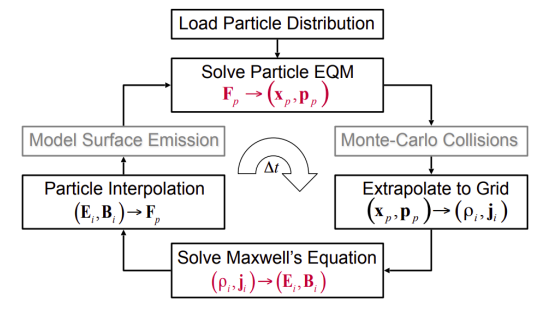
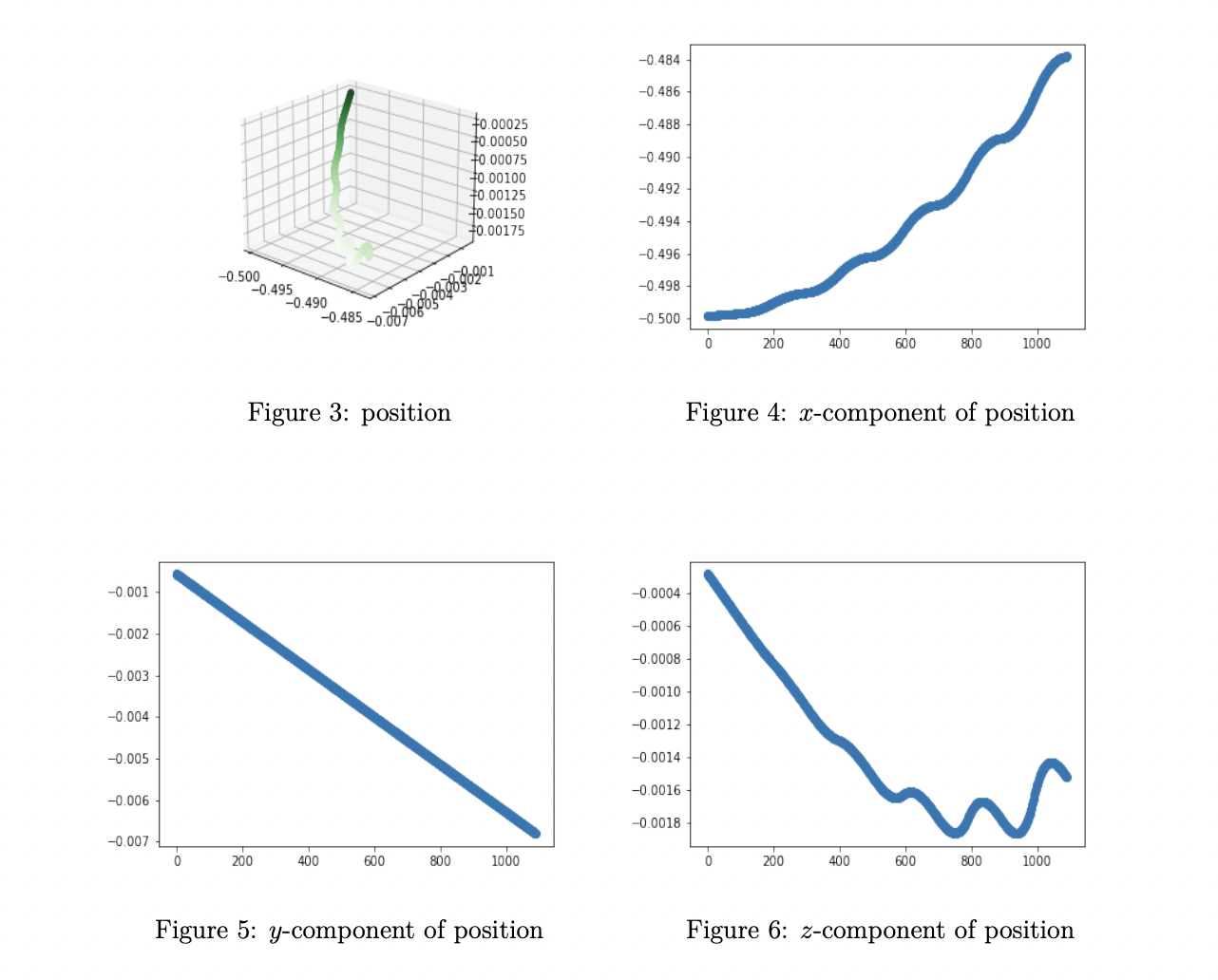


Fig 2: Simulation Flow Chart for PIC methods

**41**



## 

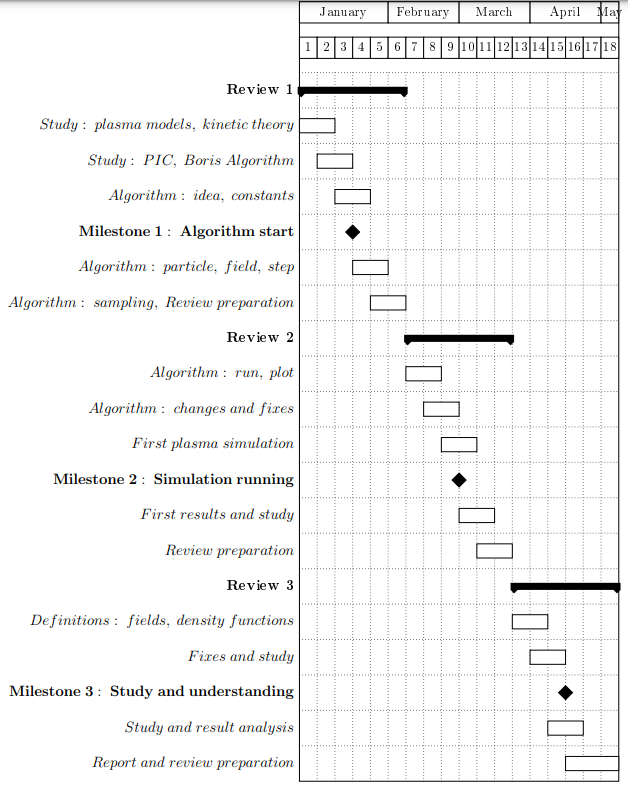
## 

## 42

## 

**43**

**List of Tables**

**Gantt Chart:**

## 

## 44

## References

[1] Chen, F. F., “**Introduction to plasma physics and controlled fusion**”, New York: Plenum press, 1984. - [6] Myers, A., Colella, P., & Straalen, B. V. (2017).

[2] Myers, A., Colella, P., & Straalen, B. V. (2017)., “**A 4th-Order Particle-in-Cell Method with Phase-Space Remapping for the Vlasov–Poisson Equation**” ,SIAM Journal on Scientific Computing, 39(3), B467-B485.

[3] Qin, H., Zhang, S., Xiao, J., & Tang, W. M. (April, 2013), “**Why is Boris algorithm so good?**” .Princeton Plasma Physics Laboratory, PPPL-4872.

**Other References**

[1]Na, Yong-Su (2017), “**Introduction to nuclear fusion**” (Lecture 9 Mirror, lecture slide). SeoulNational University Open Courseware.  
[2] Föreläsning (2009), “**Charged particle motion in magnetic field**” , (lecture slide). Luleä Uni-versity of Technology.   
[3] Professor Sitaram Dash. (Fall Semester 2021)., “**MEE4005 Surface Engineering**” (lecturenotes). SMEC, VIT Vellore.  
[4] Matthew W. Kunz. (November 9, 2020), “ **Introduction to Plasma Astrophysics**” (lecturenotes). Princeton Plasma Physics Laboratory. - we learned about introductory concepts on plasma

[5] 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

**45**