

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.



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)


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.

 18BEM0145

 18BME2109

 18BME2104

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 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

	Page No.
Acknowledgement	i
Executive Summary	ii
Table of Contents	lii
List of Figures	ix
List of Tables	xiv
Abbreviations	xvi
Symbols and Notations	xix
1 INTRODUCTION	1
1.1 Objective	1
1.2 Motivation	1
1.3 Background	1
2 PROJECT DESCRIPTION AND GOALS	1
3 TECHNICAL SPECIFICATION	2
4 DESIGN APPROACH AND DETAILS (as applicable)	4
4.1 Design Approach / Materials & Methods	5
4.2 Codes and Standards	6
4.3 Constraints, Alternatives and Tradeoffs	8
5 SCHEDULE, TASKS AND MILESTONES	9
6 PROJECT DEMONSTRATION	11
7 RESULT & DISCUSSION	13
8 SUMMARY	40
9 REFERENCES	45

APPENDIX A

List of Figures

Figure No.	Title	Page No.
1	Magnetron Sputtering	41
2	Simulation Flow Chart	41
3	Position plot	42
4	X Position	42
5	Y Position	42
6	Z position	42
7	Velocity	43
8	X Velocity	43
9	Y Velocity	43
10	Z Velocity	43
11	X position	44
12	Y position	44
13	Z position	
14		

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.

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

$$\frac{d\mathbf{v}}{dt} = \frac{q}{m} (\mathbf{E} + \mathbf{v} \times \mathbf{B})$$

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 $f(x, y, z, v_x, v_y, v_z, t)$ which describes the number of particles at position (x, y, z) at time t with velocities between v_x and $v_x + dv_x$, v_y and $v_y + dv_y$, v_z and $v_z + dv_z$ in directions x , y and z respectively. For a simpler notation $f(x, y, z, v_x, v_y, v_z, t)$ is denoted as $f(\mathbf{x}, \mathbf{v}, t)$. The expression

$$\int_{all\ v_x} dv_x \int_{all\ v_y} dv_y \int_{all\ v_z} dv_z f(\mathbf{x}, \mathbf{v}, t)$$

gives the number of particles at position \mathbf{x} , at time t . For a simple choice of notation, it is often written as

$$\int_{all\ \mathbf{v}} d^3v f(\mathbf{x}, \mathbf{v}, t) \quad or \quad \int_{all\ \mathbf{v}} d\mathbf{v} f(\mathbf{x}, \mathbf{v}, t)$$

A density function is said to be normalized if

$$\int_{all\ \mathbf{v}} d\mathbf{v} f(\mathbf{x}, \mathbf{v}, t) = 1$$

Such a density function is also denoted with a hat as $\hat{f}(\mathbf{x}, \mathbf{v}, t)$.

The average velocity \bar{v} for a density function $\hat{f}(\mathbf{x}, \mathbf{v}, t)$ is calculated as

$$\int_{all} d\mathbf{v} \mathbf{v} f(\mathbf{x}, \mathbf{v}, t)$$

Various other features of the distribution are: the Root Mean Square velocity v_{rms} , the average absolute velocity $|\mathbf{v}|$, the average velocity in z direction $|v_z|$, etc. The evolution of the particles is described as the changing of the density function. Boltzmann equation is often used to describe this dynamics:

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f + \frac{\mathbf{F}}{m} \cdot \partial_{\mathbf{v}} f = \left(\frac{\partial f}{\partial t} \right)_c$$

While it may look complicated at first, the left hand side of the equation is actually just a short form for:

$$\frac{df}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial x} v_x + \frac{\partial f}{\partial v_x} a_x + \frac{\partial f}{\partial y} v_y + \frac{\partial f}{\partial v_y} a_y + \frac{\partial f}{\partial z} v_z + \frac{\partial f}{\partial v_z} a_z$$

For simplicity, let us look at the 1-dimensional (2-phase dimensional) version of this expression

$$\frac{df}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial v_x} \frac{dv_x}{dt}$$

If the right hand side of the Boltzmann equation were zero, in 1-dimension (2-phase dimensions), it would read:

$$\frac{df}{dt} = 0$$

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 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:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0$$

and the Cauchy momentum equation:

$$\rho \left(\frac{\partial}{\partial t} + \mathbf{v} \cdot \nabla \right) \mathbf{v} = \mathbf{J} \times \mathbf{B} - \nabla p$$

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.

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:

$$\frac{\partial \mathbf{B}}{\partial t} = -\nabla \times \mathbf{E}$$

and the Ampere's law:

$$\mu_0 \mathbf{J} = \nabla \times \mathbf{B}$$

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

$$\frac{d\mathbf{v}}{dt} = \frac{q}{m} (\mathbf{E} + \mathbf{v} \times \mathbf{B})$$

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

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}$$

These equations are discretized to obtain

$$\frac{\mathbf{v}_{k+1} - \mathbf{v}_k}{\Delta t} = \frac{q}{m} \left[\mathbf{E}_k + \frac{(\mathbf{v}_{k+1} + \mathbf{v}_k)}{2} \times \mathbf{B}_k \right]$$

from the Lorentz Force equation, and

$$\frac{\mathbf{x}_{k+1} - \mathbf{x}_k}{\Delta t} = \mathbf{v}_{k+1}$$

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.

$$\frac{\mathbf{v}^- - \mathbf{v}_k}{(\Delta t/2)} = \frac{q}{m} \mathbf{E}_k \quad \text{or} \quad \frac{\mathbf{v}^- - \mathbf{v}_k}{\Delta t} = \frac{1}{2} \frac{q}{m} \mathbf{E}_k$$

which is often called the first half of the electric pulse.

$$\frac{\mathbf{v}^+ - \mathbf{v}^-}{\Delta t} = \frac{q}{m} \left(\frac{\mathbf{v}^+ + \mathbf{v}^-}{2} \right) \times \mathbf{B}_k$$

which is often called rotation by the magnetic field.

$$\frac{\mathbf{v}_{k+1} - \mathbf{v}^+}{(\Delta t/2)} = \frac{q}{m} \mathbf{E}_k \quad \text{or} \quad \frac{\mathbf{v}_{k+1} - \mathbf{v}^+}{\Delta t} = \frac{1}{2} \frac{q}{m} \mathbf{E}_k$$

which is almost the discretized Lorentz equation except that $(\mathbf{v}^+ + \mathbf{v}^-)$ is substituted for $(\mathbf{v}_{k+1} + \mathbf{v}_k)$. However, subtracting the first electric pulse equation from equation gives $(\mathbf{v}^+ + \mathbf{v}^-) = (\mathbf{v}_{k+1} + \mathbf{v}_k)$, 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

$$\begin{aligned} \mathbf{v}^- &= \mathbf{v}_k + q' \mathbf{E}_k \\ \mathbf{v}^+ &= \mathbf{v}^- + 2q' (\mathbf{v}^- \times \mathbf{B}_k) \\ \mathbf{v}_{k+1} &= \mathbf{v}^+ + q' \mathbf{E}_k \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \Delta t \mathbf{v}_{k+1} \end{aligned} \quad (8)$$

along with discretized position equation where $q' = \frac{q}{m} \frac{\Delta t}{2}$. 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

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

$$\widehat{f}_M := \hat{f}(\mathbf{x}, \mathbf{v}, t) = \left(\frac{m}{2\pi KT} \right)^{\frac{3}{2}} \exp \left(-\frac{v^2}{v_{th}^2} \right)$$

Where

$$v_{th}^2 = \frac{2KT}{m}$$

Some features of the Maxwellian are:

$$v_{rms} = \sqrt{\frac{3KT}{m}}, |\bar{\mathbf{v}}| = 2\sqrt{\frac{2KT}{\pi m}}, |\bar{v}_z| = \sqrt{\frac{2KT}{\pi m}}, \bar{v}_z = 0$$

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.

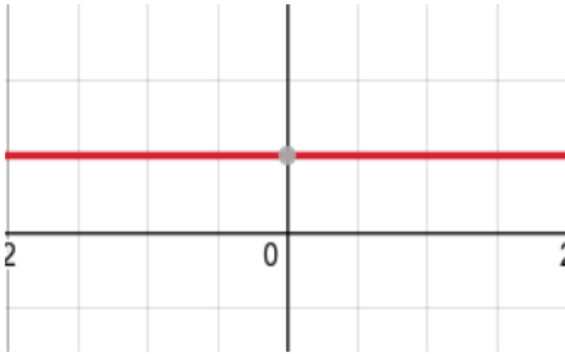


Figure 3: f_0

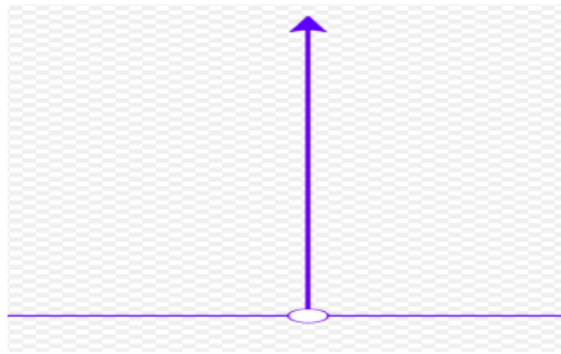


Figure 4: f_δ

Functions that look like f_0 might define the uniform distribution while functions like f_δ might describe the Dirac delta distribution. The figure for f_0 was generated using desmos graphing calculator while the figure for f_δ 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:

$$f_1 = \begin{cases} 1 - \frac{v^2}{2v_a^2} & -v_a \leq v \leq v_a \\ 0 & \text{else} \end{cases}$$

$$f_2 = \begin{cases} 1 + \frac{v^2}{2v_a^2} & v_a \leq v \leq v_a \\ 0 & \text{else} \end{cases}$$

These functions can be normalized as

$$\int_{v_x=-v_a}^{v_x=v_a} dv_x \int_{v_y=-v_a}^{v_y=v_a} dv_y \int_{v_z=-v_a}^{v_z=v_a} dv_z c_0 \left(1 - \frac{v_x^2 + v_y^2 + v_z^2}{2v_a^2} \right) = 1$$

$$\int_{v_x=-v_a}^{v_x=v_a} dv_x \int_{v_y=-v_a}^{v_y=v_a} dv_y \int_{v_z=-v_a}^{v_z=v_a} dv_z c_0 \left(1 + \frac{v_x^2 + v_y^2 + v_z^2}{2v_a^2} \right) = 1$$

$$\hat{f}_1 = \begin{cases} \frac{1}{4v_a^3} \left(1 - \frac{v^2}{2v_a^2} \right) & -v_a \leq v \leq v_a \\ 0 & \text{else} \end{cases}$$

$$\hat{f}_2 = \begin{cases} \frac{1}{12v_a^3} \left(1 + \frac{v^2}{2v_a^2} \right) & v_a \leq v \leq v_a \\ 0 & \text{else} \end{cases}$$

The graph of these functions (considering v as a single variable) were generated using desmos graphing calculator. The figures represent \hat{f}_1 and \hat{f}_2 extrapolated to beyond the defined range

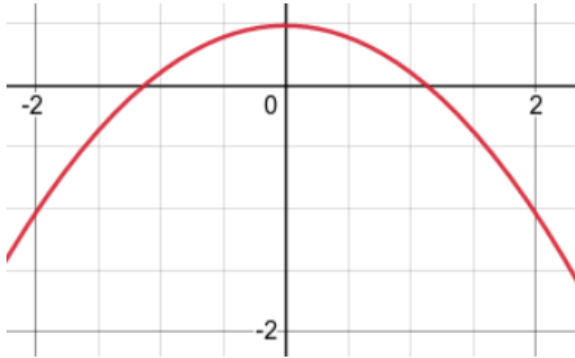


Figure 5: \hat{f}_1

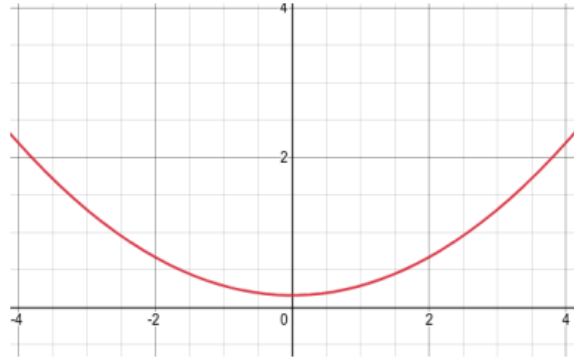


Figure 6: \hat{f}_2

In the upper half of the plane, \hat{f}_1 looks approximately like a Gaussian distribution which the Maxwellian is for a distribution of velocity (not speed) for a fixed temperature. \hat{f}_2 is sort of the opposite of \hat{f}_1 . 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.

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.

Required functionalities and Files:

1. Constants - **constants.ipynb**
2. Particle - **particle.ipynb**
3. Electric and Magnetic fields - **field.ipynb**
4. Particle initialization - **sampling.ipynb**
5. Updating the particles - **step.ipynb**
6. Batches of updates - **run.ipynb**
7. Plotting - **plot.ipynb**

Constants - constants.ipynb

The **constants.ipynb** notebook describes some constants useful in the program. Some useful constants are e (electron charge) m_e (electron mass), charges and masses of ions in the plasma, a.m.u (atomic mass unit), N_A (Avogadro's number), ϵ_0 (permittivity of vacuum), μ_0 (permeability of vacuum), K or k_B (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.

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 `run.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.

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

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.

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

pair of capacitor plates, if there is a particle at the same z-coordinate as that point. So currently we use the expression $E = V \cdot dr$ 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.

$$\widehat{f}_M := \widehat{f}(\mathbf{x}, \mathbf{v}, t) = \left(\frac{m}{2\pi KT}\right)^{\frac{3}{2}} \exp\left(-\frac{\mathbf{v}^2}{v_{th}^2}\right) \quad (1)$$

where

$$v_{th}^2 = \frac{2KT}{m}$$

For our convenience, we use the density function with speed instead of velocity which is defined as:

$$\widehat{f}_m := \widehat{f}(\mathbf{x}, v, t) = \left(\frac{m}{2\pi KT}\right)^{\frac{3}{2}} 4\pi v^2 \exp\left(-\frac{v^2}{v_{th}^2}\right) \quad (2)$$

which is obtained by integrating over the solid angle in the velocity variable.

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

$$\begin{aligned} \langle v \rangle &= \int_{v=-\infty}^{v=\infty} dv \, v \, \widehat{f}_m = \int_{v=-\infty}^{v=\infty} dv \, v \, \left(\frac{m}{2\pi KT}\right)^{\frac{3}{2}} 4\pi v^2 \exp\left(-\frac{v^2}{v_{th}^2}\right) \\ &= 4\pi \left(\frac{m}{2\pi KT}\right)^{\frac{3}{2}} \int_{v=-\infty}^{v=\infty} dv \, v^3 \exp\left(-\frac{v^2}{v_{th}^2}\right) \\ &= 4\pi \left(\frac{m}{2\pi KT}\right)^{\frac{3}{2}} \frac{v_{th}^4}{2} = 4\pi \left(\frac{1}{\pi v_{th}^2}\right)^{\frac{3}{2}} \frac{v_{th}^4}{2} \\ &= \frac{2}{\sqrt{\pi}} v_{th} = \frac{2}{\sqrt{\pi}} \sqrt{\frac{2KT}{m}} = \frac{2}{\sqrt{\pi}} \sqrt{\frac{2RT}{M}} \end{aligned}$$

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

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

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

```

1 # c1Maxwell means checking the Maxwellian sampling
2 c1Maxwell = Run()
3 # Create 100 particles based on the data available in the files
4 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

```
1 # Take the 0th batch of particles
2 c1Maxwell_batch = c1Maxwell.batches[0]['H ions']
3 # Take the initial positions and velocities of the particles
4 c1Maxwell_positions = [] c1Maxwell_velocities
5 = []
6 for particle in c1Maxwell_batch.particles:
7     c1Maxwell_positions.append(particle.r)
8     c1Maxwell_velocities.append(particle.v)
9 # Let's now look at the velocities
10 c1Maxwell_velocities
11 # We need to check if they are really Maxwellian distributed #
12 Get the speeds
13 c1Maxwell_speeds = np.sqrt( [ (c1Maxwell_velocities[i][0] ** 2) +
14     ↪ (c1Maxwell_velocities[i][1] ** 2) + (c1Maxwell_velocities[i][2] ** 2) for i in
15     ↪ range(len(c1Maxwell_velocities)) ] )
16 c1Maxwell_meanspeed = np.sum(c1Maxwell_speeds) / c1Maxwell_speeds.size
```

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.

Initialization

```
1 # Create 100 particles based on the sampled velocities of H2 gas
2 # We consider just for this test case that a Hydrogen molecules to be sampled with a
  ↳ Maxwellian distribution
3 # We create a new batch on the same Run instance
4 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']
1 c1Maxwell_positions_batch2 = []
2 c1Maxwell_velocities_batch2 = []
3 for particle in c1Maxwell_batch2.particles:
4 c1Maxwell_positions_batch2.append(particle.r)
5 c1Maxwell_velocities_batch2.append(particle.v)
6 c1Maxwell_speeds_batch2 = np.sqrt( [ (c1Maxwell_velocities_batch2[i][0] ** 2) +
7   ↳ (c1Maxwell_velocities_batch2[i][1] ** 2) + (c1Maxwell_velocities_batch2[i][2] ** 2)
8   ↳ for i in range(len(c1Maxwell_velocities_batch2)) ] )
c1Maxwell_meanspeed_batch2 = np.sum(c1Maxwell_speeds_batch2) /
9   ↳ c1Maxwell_speeds_batch2.size
```

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

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

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.

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:

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

where $q' = \frac{q}{m} \frac{\Delta t}{2}$

Initialization

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


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)  
2  
3 #Let's inspect the positions and velocities of the particles at index 1 and 7  
4 c2Boris_p1 = c2Boris_positions_and_velocities[1]  
5 c2Boris_p7 = c2Boris_positions_and_velocities[7]  
6  
7 #Let's take the positions and velocities of the particle 1 after the 3rd and the 4th  
8 ↳ time steps.  
9 c2Boris_p134_p = [c2Boris_p1[3][1], c2Boris_p1[4][1]]  
10 c2Boris_p134_v = [c2Boris_p1[3][2], c2Boris_p1[4][2]]  
11  
12 #Similary for particle 7  
13 c2Boris_p734_p = [c2Boris_p7[3][1], c2Boris_p7[4][1]]  
  c2Boris_p734_v = [c2Boris_p7[3][2], c2Boris_p7[4][2]]
```

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

[39771. 83801956555, -5029. 491038, 4533. 343932509505] ms⁻¹

and after the 4th update, it changed to:

[48909. 86581773698, -5029. 491038, 8798. 399243869582] ms⁻¹

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

[-0. 48985112694809785, -0. 0020117964152, 0. 00017005183797547688] m

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

[-0. 4849601403663242, -0. 0025147455189999997, 0. 001049891762362435] m

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

$$q' = \frac{q}{m} \frac{\Delta t}{2} = \frac{1.602176634 \times 10^{-19} \text{ C}}{1.008 \text{ a.m.u.} \times 1.6605390666 \times 10^{-27} \text{ kg a.m.u.}^{-1}} \frac{10^{-7} \text{ s}}{2} = 4.78597877761177 \text{ Cskg}^{-1}$$

$$\mathbf{v}^- = \mathbf{v}_3 + q' \mathbf{E} = \begin{bmatrix} 39771.83801956555 \\ -5029.491038 \\ 4533.343932509505 \end{bmatrix} + 4.78597877761177 \begin{bmatrix} 1000 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 44557.81679718 \\ -5029.491038 \\ 4533.34393251 \end{bmatrix}$$

$$\begin{aligned} \mathbf{v}^+ = \mathbf{v}^- + 2q' (\mathbf{v}^- \times \mathbf{B}) &= \begin{bmatrix} 44557.81679718 \\ -5029.491038 \\ 4533.34393251 \end{bmatrix} + 2 \cdot 4.78597877761177 \left(\begin{bmatrix} 44557.81679718 \\ -5029.491038 \\ 4533.34393251 \end{bmatrix} \times \begin{bmatrix} 0.0 \\ 0.01 \\ 0.0 \end{bmatrix} \right) \\ &= \begin{bmatrix} 44123.88704013 \\ -5029.491038 \\ 8798.39924387 \end{bmatrix} \end{aligned}$$

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

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

We see that the values for \mathbf{x}_3 , \mathbf{x}_4 , \mathbf{v}_3 and \mathbf{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 3rd step of the update, the velocity of the particle was:

$$[38895.85871094631, -8670.854777, 13914.969423620274] \text{ ms}^{-1}$$

and after the 4th update, it changed to:

$$[47135.881299118584, -8670.854777, 18096.176367366777] \text{ ms}^{-1}$$

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:

$$[-0.48495338711336006, -0.0043354273885, 0.0056971673271299424]$$

$$\mathbf{v}^- = \mathbf{v}_3 + q'\mathbf{E} = \begin{bmatrix} 38895.85871095 \\ -8670.854777 \\ 13914.96942362 \end{bmatrix} + 4.78597877761177 \begin{bmatrix} 1000 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 43681.83748856 \\ -8670.854777 \\ 13914.96942362 \end{bmatrix}$$

$$\mathbf{v}^+ = \mathbf{v}^- + 2q'(\mathbf{v}^- \times \mathbf{B}) = \begin{bmatrix} 43681.83748856 \\ -8670.854777 \\ 13914.96942362 \end{bmatrix} + 2 \cdot 4.78597877761177 \left(\begin{bmatrix} 43681.83748856 \\ -8670.854777 \\ 13914.96942362 \end{bmatrix} \times \begin{bmatrix} 0.0 \\ 0.01 \\ 0.0 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 42349.90252151 \\ -8670.854777 \\ 18096.17636737 \end{bmatrix}$$

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

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

We see that the values for \mathbf{x}_3 , \mathbf{x}_4 , \mathbf{v}_3 and \mathbf{v}_4 for particle 7 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.

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.

```

1 #Create a constants object instance to access the constants from constants.ipynb file
2 constants = Constants()
3 # Create a run object instance
4 s1 = Run()
5 # Create 100 Hydrogen ions whose:
6 # speeds are Maxwellian sampled, velocity directions are uniform randomly sampled # positions
7 # are all sampled such that particles start at [-0.5, 0, 0]
8 # 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)

```

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)):
24     desc = 'E_scale = ' + str(E_scale[i]) + ' ' + 'B_scale = ' + str(B_scale[i])
25     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)
26     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.

```

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
2 s1_descE_is_1_p0_ps_fig = plt.figure()
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,
12     ↪ s1_descE_is_1_p0_ps_zdata,
13     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,
    ↪ s1_descE_is_n1_p0_ps_zdata, c=s1_descE_is_n1_p0_ps_zdata, cmap='Greens');
27
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.

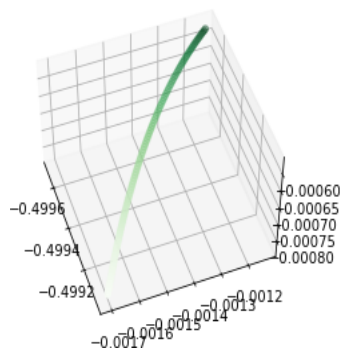


Figure 1: Electric field scaled by 1

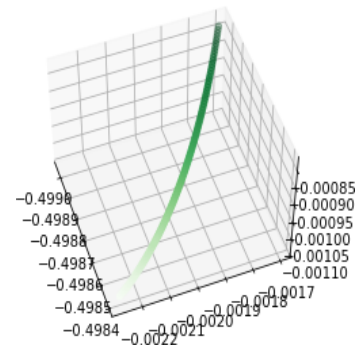


Figure 2: Electric field scaled by -1

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
2  s1_allfieldkeys = list(s1_batch_ps_and_vs.keys())
3  s1_allfield_p0_ps = []
4  s1_allfield_p0_vs = []
5  for akey in s1_allfieldkeys:
6      s1_histories = s1_batch_ps_and_vs[akey]
7
8
9  #Take particle 0
10 s1_p0 = s1_histories[0]
11
12 #Take ps and vs
13 for i in range(len(s1_p0)):
14     s1_allfield_p0_ps.append(s1_p0[i][1])
15     s1_allfield_p0_vs.append(s1_p0[i][2]) s1_allfield_p0_ps
16     = np.array(s1_allfield_p0_ps) s1_allfield_p0_vs =
17     np.array(s1_allfield_p0_vs)
18
19
20 #Plot the position
21 s1_allfield_p0_ps_fig = plt.figure()
22 s1_allfield_p0_ps_ax = plt.axes(projection='3d')
23 s1_allfield_p0_ps_ax.view_init(20, -50)

```

Data for three-dimensional scattered points

```

24 # for position update history of particle 0 during all field configurations
25 s1_allfield_p0_ps_zdata = [elem[2] for elem in s1_allfield_p0_ps]
26 s1_allfield_p0_ps_xdata = [elem[0] for elem in s1_allfield_p0_ps]
27 s1_allfield_p0_ps_ydata = [elem[1] for elem in s1_allfield_p0_ps]
28 s1_allfield_p0_ps_ax.scatter3D(s1_allfield_p0_ps_xdata, s1_allfield_p0_ps_ydata,
↪ s1_allfield_p0_ps_zdata,
29 c=s1_allfield_p0_ps_zdata, cmap='Greens');
30 plt.savefig('ps1', dpi='figure', format='png')

```

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

```

1 # Plot x position
2 s1_allfield_p0_ps_x_fig = plt.figure()
3 s1_allfield_p0_ps_x_ax = plt.axes()
4 s1_allfield_p0_ps_x_ax.scatter(np.arange(len(s1_allfield_p0_ps_xdata)),
↪ s1_allfield_p0_ps_xdata); plt.savefig('psx1',
5 dpi='figure', format='png')
6
7
8 # Plot y position
9 s1_allfield_p0_ps_y_fig = plt.figure()
10 s1_allfield_p0_ps_y_ax = plt.axes()
11 s1_allfield_p0_ps_y_ax.scatter(np.arange(len(s1_allfield_p0_ps_ydata)),
↪ s1_allfield_p0_ps_ydata); plt.savefig('psy1',
12 dpi='figure', format='png')
13
14
15 # Plot z position
16 s1_allfield_p0_ps_z_fig = plt.figure()
17 s1_allfield_p0_ps_z_ax = plt.axes()
18 s1_allfield_p0_ps_z_ax.scatter(np.arange(len(s1_allfield_p0_ps_zdata)),
↪ s1_allfield_p0_ps_zdata); plt.savefig('psz1',
19 dpi='figure', format='png')

```

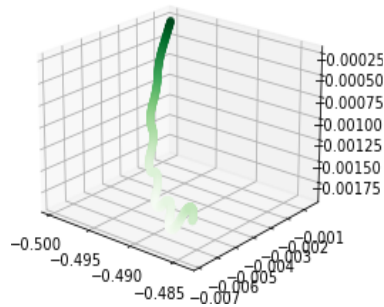



Figure 3: position

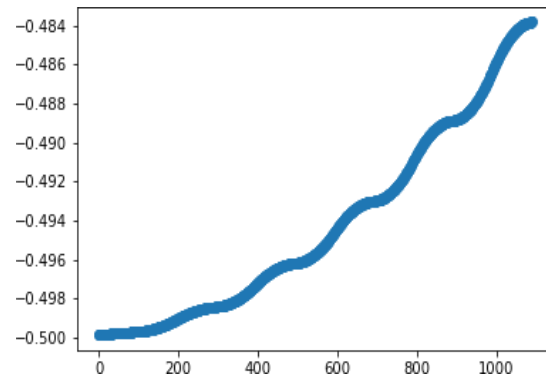


Figure 4: x -component of position

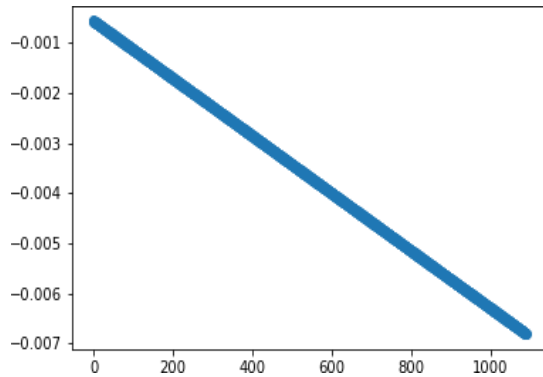


Figure 5: y -component of position

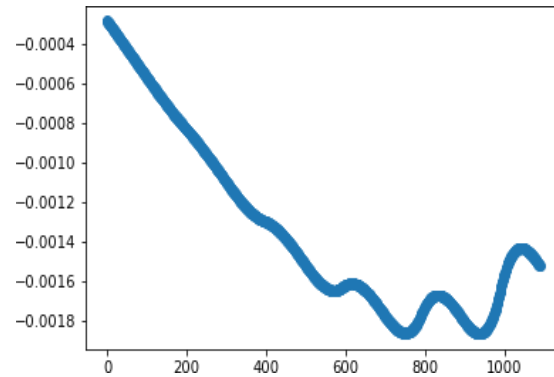


Figure 6: z -component of position

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

```

1  # Plot the velocity
2  s1_allfield_p0_vs_fig = plt.figure()
3  s1_allfield_p0_vs_ax = plt.axes(projection='3d')
4  s1_allfield_p0_vs_ax.view_init(20, -80)
5
6  # Data for three-dimensional scattered points
7  # for position update history of particle 0 during all field configurations
8  s1_allfield_p0_vs_zdata = [elem[2] for elem in s1_allfield_p0_vs] # Animate this plot as
9  ↵ well.
10 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]

```

```

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.

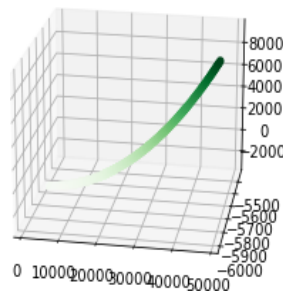


Figure 7: velocity

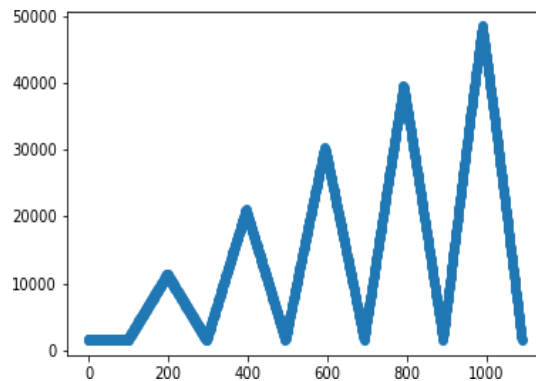


Figure 8: x -component of velocity

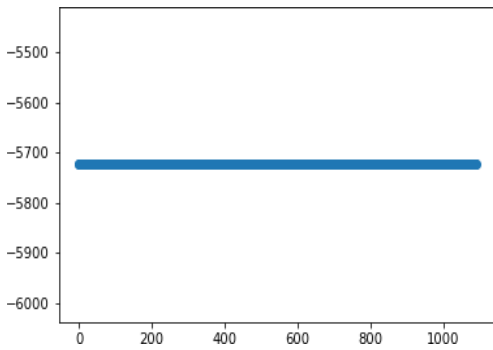


Figure 9: y-component of velocity

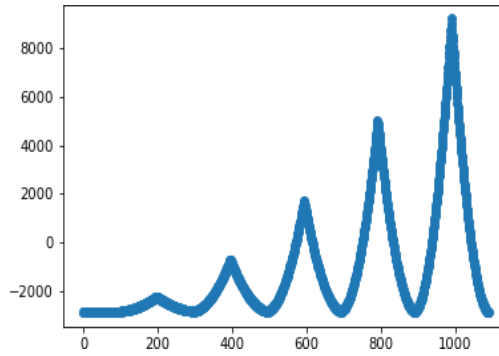


Figure 10: z-component of velocity

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 figure(3) and figure(6), although we also see the effect of particle having a non- zero initial velocity in some direction. What we see in figure(3) which is perhaps more clear in figure(6) 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 figure(7) 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) and (10) help us understand this better. We also observe in figure (9) 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.

Animations were generated using the following parts of the program.

```
1 def plot_animation_3d(positions):
2     '''
3     This function can plot both positions and velocities
4     '''
5
6     FRAMES = np.shape(positions)[0]
7     # Here positions has shape (number of particles, 3) where each entry is a
8     ↪ position which is an array of x, y, z coordinates
9     fig = plt.figure()
10    ax = fig.add_subplot(111, projection='3d')
11
12    def init():
13        ax.view_init(elev=10., azimuth=0)
14        ax.set_xlabel('x') ax.set_ylabel('y')
15        ax.set_zlabel('z')
16
17    # animation function. This is called sequentially
18    def animate(i):
19        current_index = int(positions.shape[0] / FRAMES * i) ax.cla()
20        ax.view_init(elev=10., azimuth=i)
21        ax.set_xlabel('x') ax.set_ylabel('y')
22        ax.set_zlabel('z')
23        # For line plot uncomment the following line
24        # ax.plot3D(positions[:current_index, 0], positions[:current_index, 1],
25        ↪ positions[:current_index, 2])
26        ax.scatter3D(positions[:current_index, 0], positions[:current_index, 1],
27        ↪ positions[:current_index, 2])
28
29        # call the animator.
30        anim = animation.FuncAnimation(fig, animate, init_func=init,
31        ↪ frames=FRAMES, interval=100)
32
33    return anim
```

```

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
43     ↪ position which is an array of x, y, z coordinates
44     fig = plt.figure()
45     ax = fig.add_subplot(111)
46
47     def init():
48         ax.set_xlabel('step') ax.set_ylabel(chr(include
49         + 120))
50
51     # animation function. This is called sequentially
52     def animate(i):
53         current_index = int(positions.shape[0] / FRAMES * i) ax.cla()
54         ax.set_xlabel('step') ax.set_ylabel(chr(include
55         + 120))
56         # For line plot uncomment the following line
57         # ax.plot3D(positions[:current_index, 0], positions[:current_index, 1],
58         ↪ positions[:current_index, 2])
59         ax.scatter(np.arange(len(positions))[:current_index],
60         ↪ positions[:current_index, include])
61
62         # call the animator.
63         anim = animation.FuncAnimation(fig, animate, init_func=init,
        ↪ frames=FRAMES, interval=100)
64
65     return anim

```

Running the animation generation.

```
1  # animation for the position of the particle s1_allfield_p0_ps_anim =
2  plot_animation_3d(s1_allfield_p0_ps)
3  display_animation(s1_allfield_p0_ps_anim)
4  s1_allfield_p0_ps_anim.save(r'ps1.mp4')
5
6  # x positions
7  s1_allfield_p0_ps_x_anim = plot_animation_1d(s1_allfield_p0_ps, include=0)
8
9  display_animation(s1_allfield_p0_ps_x_anim)
10 s1_allfield_p0_ps_x_anim.save(r'psx1.mp4')
11
12 # y positions
13 s1_allfield_p0_ps_y_anim = plot_animation_1d(s1_allfield_p0_ps, include=1)
14 display_animation(s1_allfield_p0_ps_y_anim)
15 s1_allfield_p0_ps_y_anim.save(r'psy1.mp4')
16
17 # z positions
18 s1_allfield_p0_ps_z_anim = plot_animation_1d(s1_allfield_p0_ps, include=2)
19 display_animation(s1_allfield_p0_ps_z_anim)
20 s1_allfield_p0_ps_z_anim.save(r'psz1.mp4')
21
22 # Plot the velocity
23 s1_allfield_p0_vs_anim = plot_animation_3d(s1_allfield_p0_vs)
24 display_animation(s1_allfield_p0_vs_anim)
25 s1_allfield_p0_vs_anim.save(r'vs1.mp4')
26
27 # Plot x velocity
28 s1_allfield_p0_vs_x_anim = plot_animation_1d(s1_allfield_p0_vs, include=0)
29 display_animation(s1_allfield_p0_vs_x_anim)
30 s1_allfield_p0_vs_x_anim.save(r'vsx1.mp4')
31
32 # Plot y velocity
33 s1_allfield_p0_vs_y_anim = plot_animation_1d(s1_allfield_p0_vs, include=1)
34 display_animation(s1_allfield_p0_vs_y_anim)
35 s1_allfield_p0_vs_y_anim.save(r'vsy1.mp4')
36
37 # Plot z velocity
38 s1_allfield_p0_vs_z_anim = plot_animation_1d(s1_allfield_p0_vs, include=2)
39 display_animation(s1_allfield_p0_vs_z_anim)
40 s1_allfield_p0_vs_z_anim.save(r'vsz1.mp4')
```

In generating the animations for the plots, we took help from the blog post titled: **Charged Particle Trajectories in Electric and Magnetic Fields** ([1]).

```
1 VIDEO_TAG = """<video controls>
2 <source src="data:video/x-m4v;base64,{0}" type="video/mp4">
3 Your browser does not support the video tag.
4 </video>"""
5 def anim_to_html(anim):
6
7     if not hasattr(anim, '_encoded_video'):
8         f = NamedTemporaryFile(suffix='.mp4', delete=False)
9         anim.save(f.name, fps=20, extra_args=['-vcodec', 'libx264', '-pix_fmt',
10 ↪ 'yuv420p'])
11         f.flush()
12         video = open(f.name, "rb").read()
13         f.close()
14         anim._encoded_video = base64.b64encode(video).decode('utf-8')
15
16     return VIDEO_TAG.format(anim._encoded_video)
17 def display_animation(anim):
18     plt.close(anim._fig)
19     return HTML(anim_to_html(anim))
```

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.

```
1 s1_particles = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90] # take particles at these indices
2 s1_allfieldkeys = list(s1_batch_ps_and_vs.keys())
3 s1_allfield_10p_ps = []
4 s1_allfield_10p_vs = []
5
6
7 for aparticle in s1_particles:
8     s1_allfield_ap_ps = [] s1_allfield_ap_vs
9     = []
10    # Same procedure for as a single particle
11    for akey in s1_allfieldkeys:
12        s1_histories = s1_batch_ps_and_vs[akey]
13    #Take aparticle
14    s1_ap = s1_histories[aparticle]
15    #Take ps and vs
16    for i in range(len(s1_p0)):
17        s1_allfield_ap_ps.append(s1_ap[i][1])
18        s1_allfield_ap_vs.append(s1_ap[i][2]) s1_allfield_ap_ps
19        = np.array(s1_allfield_ap_ps) s1_allfield_ap_vs
20        np.array(s1_allfield_ap_vs)
```

```

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,
41                                     ↵ s1_allfield_ap_ps_ydata, s1_allfield_ap_ps_zdata,
42                                     ↵ c=s1_allfield_p0_ps_zdata);
43 plt.savefig('multips1', dpi='figure', format='png')
44
45 # Plot x positions
46 s1_allfield_10p_ps_x_fig = plt.figure()
47 s1_allfield_10p_ps_x_ax = plt.axes()
48 for i in range(len(s1_particles)):
49     s1_allfield_ap_ps_xdata = [elem[0] for elem in s1_allfield_10p_ps[i]]
50     s1_allfield_10p_ps_x_ax.scatter(np.arange(len(s1_allfield_ap_ps_xdata)),
51                                     ↵ s1_allfield_ap_ps_xdata);
52 plt.savefig('multipsx1', dpi='figure', format='png')
53
54 # Plot y positions
55 s1_allfield_10p_ps_y_fig = plt.figure()
56 s1_allfield_10p_ps_y_ax = plt.axes()
57 for i in range(len(s1_particles)):
58     s1_allfield_ap_ps_ydata = [elem[1] for elem in s1_allfield_10p_ps[i]]
59     s1_allfield_10p_ps_y_ax.scatter(np.arange(len(s1_allfield_ap_ps_ydata)),
60                                     ↵ s1_allfield_ap_ps_ydata);
61 plt.savefig('multipsy1', dpi='figure', format='png')
62
63 # Plot z positions
64 s1_allfield_10p_ps_z_fig = plt.figure()
65 s1_allfield_10p_ps_z_ax = plt.axes()
66 for i in range(len(s1_particles)):
67     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')

```

We get the following plots of the positions:

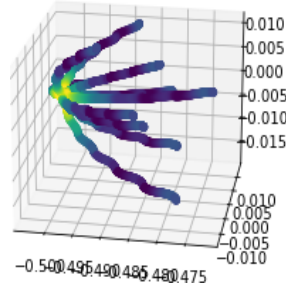


Figure 11: positions

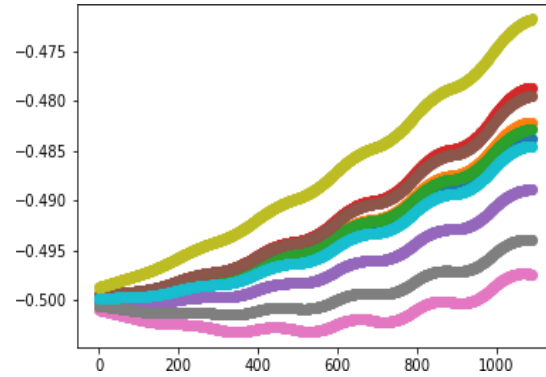


Figure 12: x -component of positions

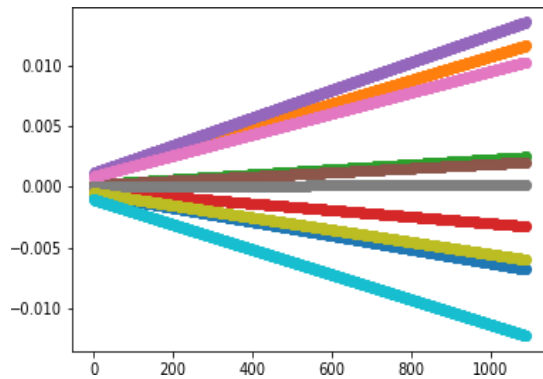


Figure 13: y -component of positions

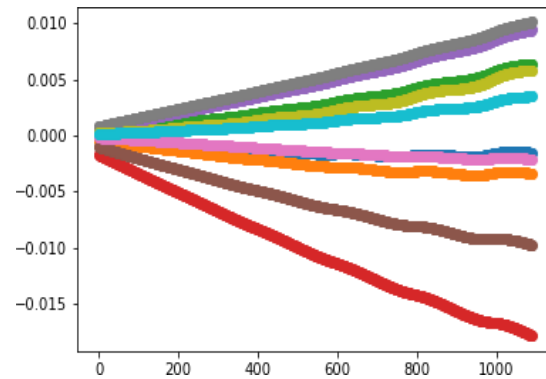


Figure 14: z -component of positions

In figure (11), we observe that all the particles begin at the same point and spread out based on their velocities. In figures (12), (13) and (14), 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.

```

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
9      ↪ this plot as well.
10     s1_allfield_ap_vs_xdata = [elem[0] for elem in s1_allfield_10p_vs[i]]
11     s1_allfield_ap_vs_ydata = [elem[1] for elem in s1_allfield_10p_vs[i]]
12     s1_allfield_10p_vs_ax.scatter3D(s1_allfield_ap_vs_xdata,
13     ↪ s1_allfield_ap_vs_ydata, s1_allfield_ap_vs_zdata, c=s1_allfield_p0_vs_zdata);
14
15     plt.savefig('multivs1', dpi='figure', format='png')
16
17     # Plot x velocities
18     s1_allfield_10p_vs_x_fig = plt.figure()
19     s1_allfield_10p_vs_x_ax = plt.axes()
20     for i in range(len(s1_particles)):
21         s1_allfield_ap_vs_xdata = [elem[0] for elem in s1_allfield_10p_vs[i]]
22         s1_allfield_10p_vs_x_ax.scatter(np.arange(len(s1_allfield_ap_vs_xdata)),
23         ↪ s1_allfield_ap_vs_xdata);
24     plt.savefig('multivsx1', dpi='figure', format='png')
25
26     # Plot y velocities
27     s1_allfield_10p_vs_y_fig = plt.figure()
28     s1_allfield_10p_vs_y_ax = plt.axes()
29     for i in range(len(s1_particles)):
30         s1_allfield_ap_vs_ydata = [elem[1] for elem in s1_allfield_10p_vs[i]]
31         s1_allfield_10p_vs_y_ax.scatter(np.arange(len(s1_allfield_ap_vs_ydata)),
32         ↪ s1_allfield_ap_vs_ydata);
33     plt.savefig('multivsy1', dpi='figure', format='png')
34
35     # Plot z velocities
36     s1_allfield_10p_vs_z_fig = plt.figure()
37     s1_allfield_10p_vs_z_ax = plt.axes()
38     for i in range(len(s1_particles)):
39         s1_allfield_ap_vs_zdata = [elem[2] for elem in s1_allfield_10p_vs[i]]
40         s1_allfield_10p_vs_z_ax.scatter(np.arange(len(s1_allfield_ap_vs_zdata)),
41         ↪ s1_allfield_ap_vs_zdata);
42     plt.savefig('multivsz1', dpi='figure', format='png')

```

We get the following plots for the velocities:

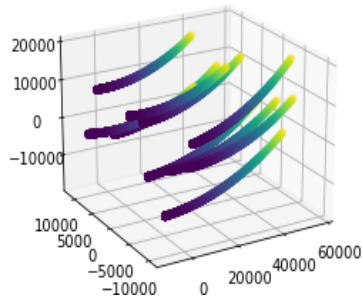


Figure 15: velocities

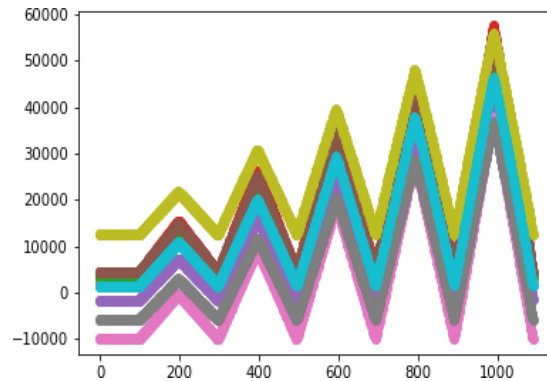


Figure 16: x-component of velocities

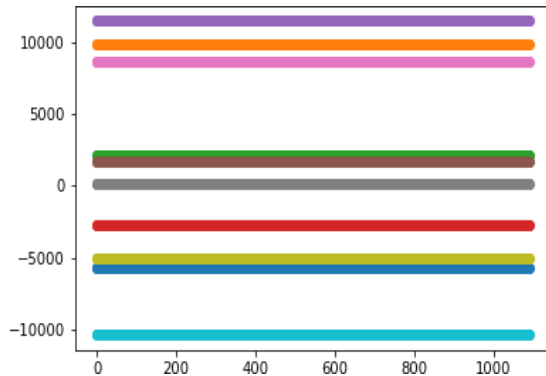


Figure 17: y-component of velocities

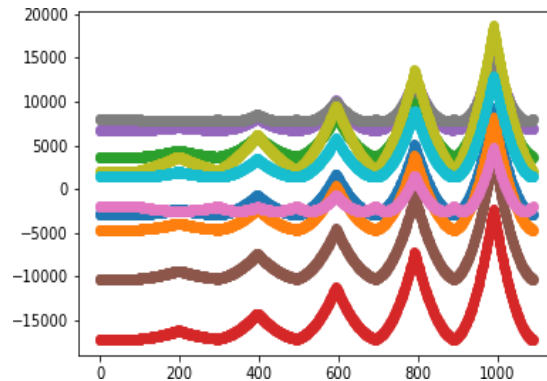


Figure 18: z-component of velocities

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) and (18) are not observed in figure (15) 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):
2     ' ' '
3     Here each element of positions is data for 1 particle that one would give as
4     input to
    plot_animation_3d function, i.e. position or velocity update history of 1
    particle
```

```

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., azimuth=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., azimuth=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],
31        ↪ positions[:current_index, 2])
32        for position in positions:
33            ax.scatter3D(position[:current_index, 0], position[:current_index, 1],
34            ↪ position[:current_index, 2])
35
36        # call the animator.
37        anim = animation.FuncAnimation(fig, animate, init_func=init,
38        ↪ frames=FRAMES, interval=100)
39
40        return anim
41
42    def multiplot_animation_1d(positions, include):
43        ''' Here each element of positions is data for 1 particle that one would give as
44        ↪ input to
45        plot_animation_3d function, i.e. position or velocity update history of 1
46        ↪ particle
47
48        This function can plot both positions and velocities

```

```

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         # ax.plot3D(positions[:current_index, 0], positions[:current_index, 1],
66         #             positions[:current_index, 2])
67         for position in positions: ax.scatter(np.arange(len(position))[:current_index],
68         #             position[:current_index, include])
69
70         # call the animator.
71         anim = animation.FuncAnimation(fig, animate, init_func=init,
72         #             frames=FRAMES, interval=100)

```

Running the animation generation.

```

1  # Animate the positions
2  s1_allfield_10p_ps_anim = multiplot_animation_3d(s1_allfield_10p_ps)
3  display_animation(s1_allfield_10p_ps_anim)
4  s1_allfield_10p_ps_anim.save(r'multips1.mp4')
5
6  # x positions
7  s1_allfield_10p_ps_x_anim = multiplot_animation_1d(s1_allfield_10p_ps, include=0)
8  display_animation(s1_allfield_10p_ps_x_anim) s1_allfield_10p_ps_x_anim.save(r'multipsx1.mp4')
9

```

```

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 $\mathbf{E} \times \mathbf{B}$ drift.

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

List of Figures

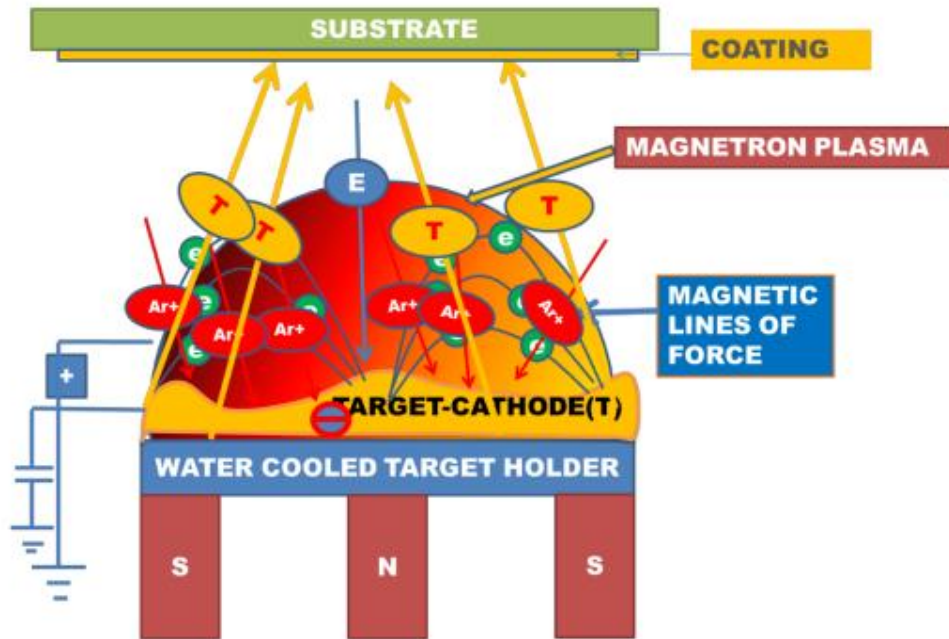


Fig 1: Magnetron Sputtering System

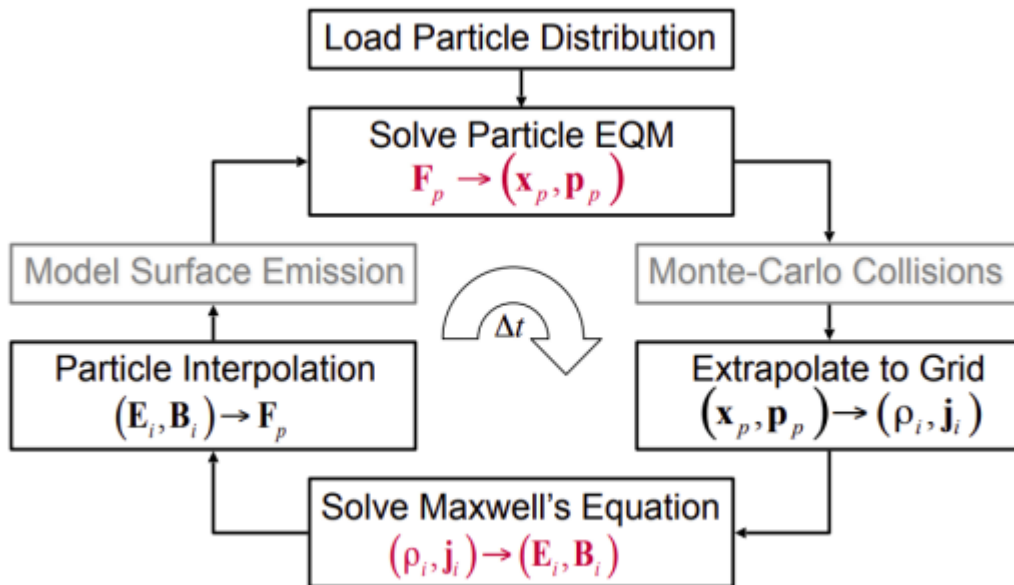


Fig 2: Simulation Flow Chart for PIC methods

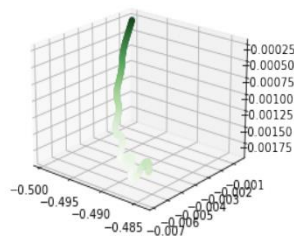


Figure 3: position

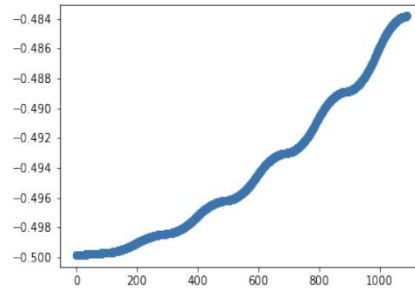


Figure 4: x -component of position

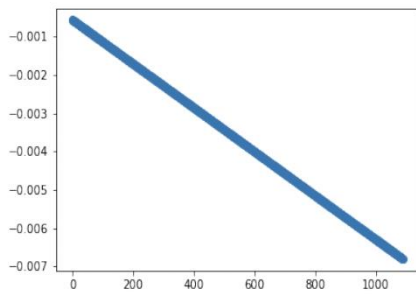


Figure 5: y -component of position

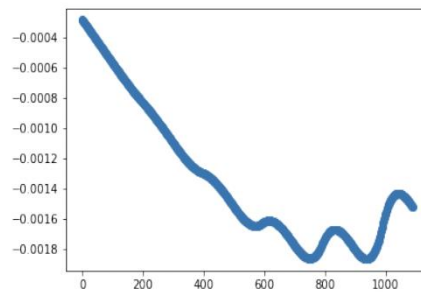


Figure 6: z -component of position

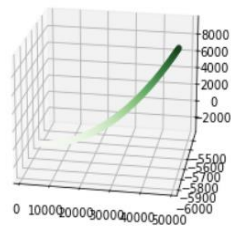


Figure 7: velocity

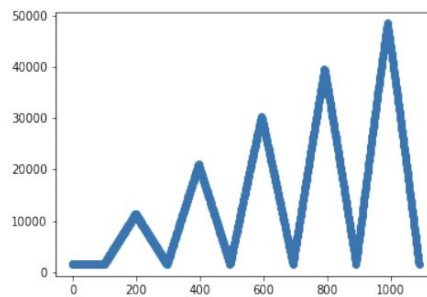


Figure 8: x -component of velocity

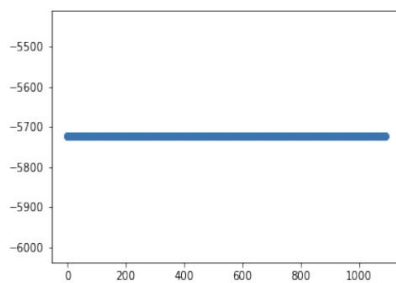


Figure 9: y -component of velocity

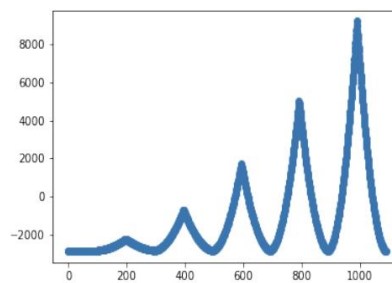


Figure 10: z -component of velocity

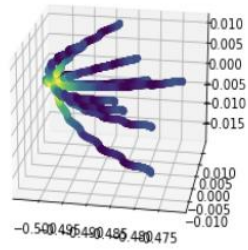


Figure 11: positions

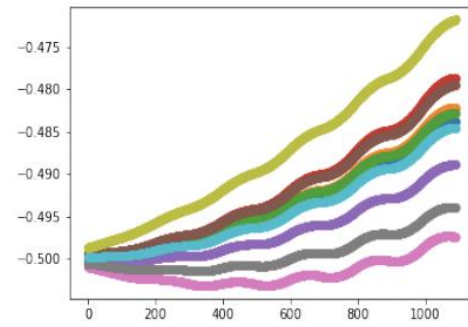


Figure 12: x -component of positions

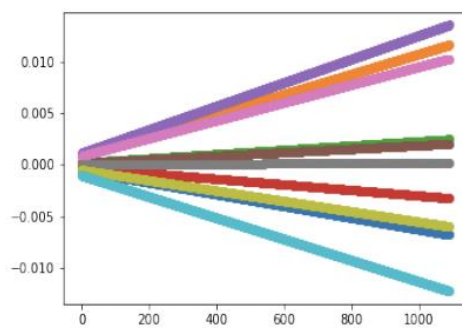


Figure 13: y -component of positions

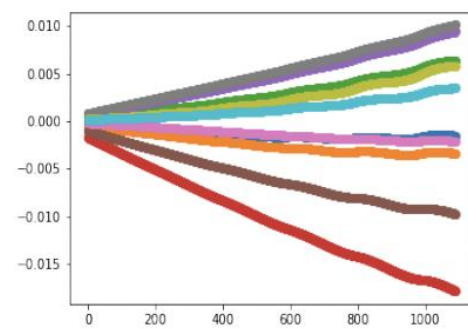


Figure 14: z -component of positions

List of Tables

Gantt Chart:



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). Seoul National University Open Courseware.
- [2] Föreläsning (2009), “**Charged particle motion in magnetic field**”, (lecture slide). Luleå University of Technology.
- [3] Professor Sitaram Dash. (Fall Semester 2021)., “**MEE4005 Surface Engineering**” (lecture notes). SMEC, VIT Vellore.
- [4] Matthew W. Kunz. (November 9, 2020), “**Introduction to Plasma Astrophysics**” (lecture notes). 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>

