

Magnetic Mirror Effect in Magnetron Plasma:

Modeling of Plasma Parameters

February 3, 2022

GitHub repository:

<https://github.com/18BME2104/MagneticMirror>

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**

1. Constants - constants.ipynb

Define constants

```
1  class Constants:
2      def __init__(self):
3          self.constants = {
4              #'symbol': ['value', 'unit', 'digits',
5                          '10 ^ power', 'number of digits', '
6                          description'],
7          }
8      def show_constant(self, symbol):
```

Useful constants:

electron charge, electron mass, ion masses, ion charges
atomic mass unit, Avogadro's number,
permittivity of vacuum, permeability of vacuum,
Boltzmann's constant

2. Particle - particle.ipynb

Define state of a particle and update it

1. Particle State:
Position and Velocity
name, mass, charge
2. Update State:
Boris Update
... other updates strategies

```
1  class Particle:
2      #... other things
3
4      def Boris_update(self, afield, argsE, argsB):
5          # Define q_prime
6          q_prime = (self.charge / self.mass) * (dt /
7              2)
8
9          # Get E and B fields from the afield
10             argument by passing in the current
11             position of the particle
12
13             argsE = V, center
14             E = afield.get_E_field(self.r, V, center)
15             argsB = n, l, R, B_hat, mu_0
16             B = afield.get_B_field(self.r, n, l, R,
17                 B_hat, mu_0)
```

```
1      #Boris velocity update
2      v_minus = self.v + q_prime * E
3      v_plus = v_minus + q_prime * 2 * np.cross(
4          v_minus, B)
5      v_new = v_plus + q_prime * E
6
7      self.v = v_new
8
9      #could have also done:
10     #self.v += (2 * q_prime) * (E + np.cross( (
11         self.v + q_prime * E), B))
12
13     #update position
14     self.r += v_new * dt
15
16     #... some other things
```

3. Electric and Magnetic fields - field.ipynb

Define field configurations

- Electric Field:
 1. Uniform Electric Field
 2. Radial Electric Field
- Magnetic Field:
 1. Uniform Magnetic Field
 2. Magnetic Field due to Helmholtz Coil
 3. Magnetic Field due to 2 Helmholtz Coils

... other field configurations:

1. Analytic field expression
2. Apparatus like coils

```
1  class Field:
2      #... something
3
4      def uniform_E_field(self, E):
5          return E
6
7      def radial_E_field(self, r, V, center =
8          [0,0,0]):
9          #Get the distance vector of the particle
10             from the electrode
11             dr = [(r[0] - center[0]), (r[1] - center
12                 [1]), 0]
13             #Get the electric field
14             E = V * dr
15             return E
```



```
1  class Field:
2      def uniform_B_field(self, B):
3          return B
4
5      def helmholtz_coil_B_field(self, n, I, R, B_hat
6          , mu_0):
7          return ( (4/5)**1.5 * ( (mu_0 * n * I) / (R
8              ) ) * B_hat)
9
10     def two_helmholtz_B_field(self, n1, I1, R1,
11         B1_hat, n2, I2, R2, B2_hat, mu_0):
12         B1 = helmholtz(self, n1, I1, R1, mu_0,
13             B1_hat)
14         B2 = helmholtz(self, n2, I2, R2, mu_0,
15             B2_hat)
16         B_hat = B1_hat + B2_hat
17         return B_hat
```

4. Particle initialization - sampling.ipynb

Sample initial positions and velocities

- Initial Position Sampling:
 1. Same position for all: eg: valve position
 2. Same distance from a point: eg: from an electrode
 - ... other distributions
- Initial Velocity Sampling:
 1. Same velocity for all: eg: injected by pump
 2. Same speed for all
 3. Speeds based on same K.E. :
eg: accelerated through same potential
 4. Maxwellian distributed speeds
 5. Maxwellian distributed velocities
 6. Parabolic distribution
 - ... other distributions
- Writing to files
csv format

```
1  class Sampler:
2      #... something
3
4      def sample_same_given_position(self, r, n):
5          positions = []
6          for i in range(n):
7              positions.append(r)
8
9          return np.array(positions)
10     def
11         sample_same_given_speed_all_random_direction
12         (self, s, n):
13             velocities = []
14             for i in range(n):
15                 velocities.append(s *
16                                     uniform_random_unit_vector())
17             return np.array(velocities)
```

```

1  class Sampler:
2      #... something
3      def
4          sample_Maxwellian_velocity_all_random_direction
            (self, v_median, K, T, m, n):
5          speeds = sample_Maxwellian_speed(self,
            v_median, K, T, m, n)
6          velocities = []
7          for i in range(n):
8              velocities.append(speeds[i] * np.array(
                uniform_random_unit_vector()))
9          return np.array(velocities)
10
11     def sample_parabolic_velocity(self):
12         pass

```

```
1  class Sampler:
2      #... something
3      def write_to_csv_file(self, ...):
4
5          # write array to csv file
6
7      def write_file_name(self, ...):
8
9          # write the file name to the list of
            available files
10
11     #... something else
```

5. Updating the particles - step.ipynb

Evolve particles under influence of fields

1. Initialize particles: hold states
 - Use initial positions and velocities directly
 - Read initial positions and velocities from saved files
2. Initialize fields: call field configurations
3. Update particles: call updates

```
1  class Step:
2
3      # ... something
4
5      ### PARTICLES INTIALIZATION SECTION
6      def initialize_particles(self, names, q_s, m_s,
7                               r_0_s, v_0_s, a_0_s, n):
8
9          for i in range(n):
10             self.particles.append( Particle(names[i]
11                                              ], q_s[i], m_s[i], r_0_s[i], v_0_s[
12                                              i], a_0_s[i] ))
13
14
15      ### FIELDS INITIALIZATION SECTION
16      def initialize_fields(self):
17          self.fields = Field()
```

```
1  ### TIME STEP SECTION
2      def update_particles(self , dt , argsE , argsB):
3
4          for particle in self.particles:
5              particle.update(self.fields , dt , argsE ,
                             argsB)
```



```
1  class Step:
2
3      # ... something
4      def read_r_or_v_file(self , index):
5          # ...
6          return rows
7
8      def reshaper(self , array_from_file):
9          # ...
10         return reshaped_array
11
12     def read_r_or_v_file_and_reshape(self , index):
13         # ...
14         return reshaped_array
15
16     # ... something else
```

6. Batches of updates - run.ipynb

Define dynamic plasma system

1. Create particles
2. Update particles
3. Update fields
4. Create new particles
5. Remove particles

```
1  class Run:
2      def create_particles(self):
3          pass
4
5      def update_particles(self):
6          pass
7
8      def remove_particles(self):
9          # This might include particles moving
10             outside the chamber (the Electric and
11             Magnetic fields or
12             # simply positions of interest) or
13             particles being absorbed for example in
14             a coating process
15
16          pass
```

```
1      def create_fields(self):  
2          pass  
3  
4      def change_fields(self):  
5          pass  
6  
7      #... something else
```

7. Plotting - plot.ipynb

Plot functionalities

References



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