Magnetic Mirror Effect in Magnetron Plasma:

Modeling of Plasma Parameters

January 27, 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

```
class Constants:
    def __init__(self):
        self.constants = {
            #'symbol': ['value', unit', 'digits', '10 ^ power', 'number of digits', 'description'],
            }
    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

```
class Particle:
    #... other things

    def Boris_update(self, afield, argsE, argsB):
        # Define q_prime
        q_prime = (self.charge / self.mass) * (dt / 2)

        # Get E and B fields from the afield argument by passing in the current position of the particle
        argsE = V, center
        E = afield.get_E_field(self.r, V, center)
        argsB = n, I, R, B_hat, mu_0
        B = afield.get_B_field(self.r, n, I, R, B_hat, mu_0)
```

```
        #Boris velocity update
        v_minus = self.v + q_prime * E
        v_plus = v_minus + q_prime * 2 * np.cross(v_minus, B)
        v_new = v_plus + q_prime * E

        self.v = v_new


        #could have also done:
        #self.v += (2 * q_prime) * (E + np.cross( (self.v + q_prime * E), B))

        #update position
        self.r += v_new * dt

    #... 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

```python
class Field:
    #... something

    def uniform_E_field(self, E):
        return E

    def radial_E_field(self, r, V, center = [0,0,0]):
        #Get the distance vector of the particle from the electrode
        dr = [(r[0] - center[0]), (r[1] - center[1]), 0]
        #Get the electric field
        E = V * dr
        return E
```

```python
class Field:
    def uniform_B_field(self, B):
        return B

    def helmholtz_coil_B_field(self, n, I, R, B_hat, mu_0):
        return ( (4/5)**1.5 * ( (mu_0 * n * I) / (R) ) * B_hat)

    def two_helmholtz_B_field(self, n1, I1, R1, B1_hat, n2, I2, R2, B2_hat, mu_0):
        B1 = helmholtz(self, n1, I1, R1, mu_0, B1_hat)
        B2 = helmholtz(self, n2, I2, R2, mu_0, B2_hat)
        B_hat = B1_hat + B2_hat
        return B_hat
```

4. Particle initialization - sampling.ipynb

Sample initial positions and velocities

- Initial Position Samping:
  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

```
class Sampler:
    #... something

    def sample_same_given_position(self, r, n):
        positions = []
        for i in range(n):
            positions.append(r)

        return np.array(positions)
    def sample_same_given_speed_all_random_direction(self, s, n):
        velocities = []
        for i in range(n):
            velocities.append(s * uniform_random_unit_vector())
        return np.array(velocities)
```

```
class Sampler:
    #... something
    def sample_Maxwellian_velocity_all_random_direction(self, v_median, K, T, m, n):
        speeds = sample_Maxwellian_speed(self, v_median, K, T, m, n)
        velocities = []
        for i in range(n):
            velocities.append(speeds[i] * np.array(uniform_random_unit_vector()))

        return np.array(velocities)

    def sample_parabolic_velocity(self):
        pass
```

```
class Sampler:
    #... something
    def write_to_csv_file(self, ...):

        # write array to csv file

    def write_file_name(self, ...):

        # write the file name to the list of available files

    #... 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

```python
class Step:

    # ... something

 ### PARTICELS INTIALIZATION SECTION
    def initialize_particles(self, names, q_s, m_s, r_0_s, v_0_s, a_0_s, n):

        for i in range(n):
            self.particles.append(Particle(names[i], q_s[i], m_s[i], r_0_s[i], v_0_s[i], a_0_s[i] ))

### FIELDS INITIALIZATION SECTION
    def initialize_fields(self):
        self.fields = Field()
```

```
### TIME STEP SECTION
    def update_particles(self, dt, argsE, argsB):

        for particle in self.particles:
            particle.update(self.fields, dt, argsE, argsB)
```

```python
class Step:

    # ... something
    def read_r_or_v_file(self, index):
        # ...
        return rows

    def reshaper(self, array_from_file):
        # ...
        return reshaped_array

    def read_r_or_v_file_and_reshape(self, index):
        # ...
        return reshaped_array

    # ... 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

```
class Run:
    def create_particles(self):
        pass

    def update_particles(self):
        pass

    def remove_particles(self):
        # This might include particles moving outside the chamber (the Electric and Magnetic fields or
        # simply positions of interest) or particles being absorbed for example in a coating process
        pass
```

```python
    def create_fields(self):
        pass

    def change_fields(self):
        pass

    #... 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.