particle

April 9, 2022

This file describes a particle in the plasma whose motion is affected by the externally applied Magnetic and Electric fields, as an instantiable object of the Particle class.

[1]:
```
import numpy as np # For computations
```

[3]:
```
class Particle:
    '''
    An object instance of the Particle class can:
    1. hold information of name, charge, mass, position, velocity and␣
↪optionally acceleration
    2. call methods to update the position and velocity based on different␣
↪strategies

    SUGGESTIONS FOR IMPROVEMENTS
    1.
    '''
    def __init__(self, name, q, m, r_0, v_0, a_0 = [0,0,0]):
        '''
        The properties of a particle are initialized.

        Arguments:
        self
        name: particle's name
        q: particle's charge
        m: particle's mass
        r_0: particle's initial position
        v_0: particle's initial velocity
        a_0: particle's initial acceleration
        It seems that acceleration is not necessary to be recorded according to␣
↪the current setup,
        so it is initialized to zero by default, requiring initial acceleration␣
↪as an optional argument.

        Returns:
        nothing
        '''
```

```python
        #description:: Type
        self.name = name
        #name:: String
        self.q = q
        #charge:: Number
        self.m = m
        #mass:: Number
        self.r = r_0
        #position:: [x:: Number, y:: Number, z:: Number]
        self.v = v_0
        #velocity:: [v_x:: Number, v_y:: Number, v_z:: Number]
        self.a = a_0
        #acceleration:: [a_x:: Number, a_y:: Numberm a_z:: Number]

    def __str__(self):
        '''
        For a particle, generate a string describing it, for use in printing.

        Arguments:
        self

        Returns:
        a formatted string describing the particle
        '''


        return f'{self.name} \n mass:{self.m} charge:{self.q} \
        \n position: ({self.r[0]}, {self.r[1]}, {self.r[2]}) \
        \n velocity: ({self.v[0]}, {self.v[1]}, {self.v[2]}) \
        \n acceleration: ({self.a[0]}, {self.a[1]}, {self.a[2]})'

    def update_acceleration(self, da):
        '''
        Currently all update is done in the Boris_update method.
        Using other update strategies may require using this function.
        '''


        pass


    def update_velocity(self, dv):
        '''
        Currently all update is done in the Boris_update method.
        Using other update strategies may require using this function.
        '''
```

```python
        pass

    def update_position(self, dr):
        '''
        Currently all update is done in the Boris_update method.
        Using other update strategies may require using this function.
        '''


        pass

    def update(self, args):
        '''
        Updates the particle based on some update strategy.
        Currently Boris_update() is used as the update strategy.
        Other update strategies could also be used.

        Arguments:
        self
        args: tuple of arguments that the currently used update strategy takes.

        For the Boris update method, the required arguments are:
        afield: an instance object of the Class Field defined in the file field.
↪ipynb
        dt: time step
        argsE: arguments required to call the currently used electric field
        argsB: arguments required to call the currently used magentic field

        Returns:
        nothing
        '''


        #unwrap the tuple args and get the arguments required for the currently␣
↪used update method
        #currently the Boris_update method is used
        afield, dt, argsE, argsB = args
        #Call the Boris_update method
        self.Boris_update(afield, dt, argsE, argsB)


    def Boris_update(self, afield, dt, argsE, argsB):
        '''
        Updates the particle state based on the Boris update strategy(Boris␣
↪Algorithm).
```

```
        Arguments:
        self
        afield: an instance object of the Class Field defined in the file field.
↪ipynb
        dt: time step

        argsE: arguments required to call the currently used electric field
        arguments required to call the currently used radial_E_field method of␣
↪the Field class,
        to get the electric field:
                Not required from arguments this one - r: the position where␣
↪the fields are required to be computed.
                Typically this is the position of a particle of interest.
        V: voltage at the electrode
        center : center equivalent position of the electrode
        a default coordinate of [0,0,0] may be used for the position of the␣
↪electrode

        argsB: arguments required to call the currently used magentic field
        arguments required to call the currently used helmholtz_coil_B_field␣
↪method of the Field class,
        to get the magnetic field:
        n: number of turns in the coil
        I: current in the coil
        R: radius of the coil
        B_hat: the direction of the magnetic field, i.e the axis of the coil
        mu_0: the constant mu_0 to be passed in using the constants in the␣
↪constants.ipynb file

        Returns:
        nothing
        '''

        # Define q_prime
        q_prime = (self.q / self.m) * (dt / 2)

        '''
        #This is for other E and B fields
        # 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)
        '''
```

```
        #Currently only uniform magnetic and electric fields are used
        E = np.array(argsE)
        B = np.array(argsB)

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

[ ]:

```

```