field

April 9, 2022

This file describes the Electric and Magnetic fields applied on the plasma. Methods of the Field class generate different Electric and Magnetic field configurations.

[12]:
```
import numpy as np # For computations
import sympy as sm # For symbolic use
import math # For mathematical calculations
```

[1]:
```
class Field:
    '''
    An instance object of the Field class can:
    1. call methods to get different Electric and Magnetic fields at different␣
 ↪points
    like the positions of particles in the setup

    SUGGESTIONS FOR IMPROVEMENTS
    '''



    def __init__(self):
        '''
        Currently an instance object of the Field class does not hold any␣
 ↪information.
        The field is generated by methods of this class.
        Other strategies taking in arguments could also be used
        '''



        pass

    def __str__(self):
        '''
        Currently an instance object of the Field class does not hold any␣
 ↪values and
        instead only calls methods from this class to return E and B fields
        at some position given to the methods as input
        '''
```

```
        pass

    def update(self):
        '''
        Currently an instance object of the Field class does not hold any␣
↪values and
        hence does not need to be updated.
        If an instance object of the Field class were to hold values of the␣
↪Electric and Magnetic fields
        they could be updated here.
        '''


        pass

    def get_E_field(self, args):
        '''
        Returns the Electric field by calling a configuration defined in one of␣
↪the other methods.
        Currently radial_E_field method is used.

        Arguments:
        self
        args: arguments to the method currently used for calculating the␣
↪Electric Field

        For the radial electric field for the electrode, the required arguments␣
↪are:
        r, V, center: arguments to the radial_E_field method

        Returns:
        The electric field [E_x, E_y, E_z], a list with 3 components
        '''

        '''
        #THIS IS TO USE radial_E_field configuration
        #unwrap the arguments from the tuple
        r, V, center = args
        #call the radial_E_field method and return the electric field that it␣
↪returns
        return self.radial_E_field(self, r, V, center)
        '''

        args = E
        return self.uniform_E_field(E)
```

```
    def get_B_field(self, args):
        '''
        Returns the Magnetic field by calling a configuration defined in one of␣
↪the other methods.
        Currently helmholtz_coil_B_field method is used.

        Arguments:
        self
        args: arguments to the method currently used for calculating the␣
↪Magnetic Field

        For the magnetic field created by a helmholtz coil, the required␣
↪arguments are:
        n, I, R, B_hat, mu_0: arguments to the helmholtz_coil_B_field method

        Returns:
        The magnetic field [B_x, B_y, B_z], a list of 3 components
        '''

        '''
        #THIS IS TO USE helmholtz_coil_B_field configuration
        #unwrap the arguments from the tuple
        n, I, R, B_hat, mu_0 = args
        #call the helmholtz_coil_B_field method and return the magentic field␣
↪that it returns
        return self.helmholtz_coil_B_field(self, n, I, R, B_hat, mu_0)
        '''

        args = B
        return self.uniform_B_field(B)

    def uniform_E_field(self, E):
        '''
        This is a particular configuration of the electric field,
        a uniform electric field

        Arguments:
        self
        E: the uniform electric field, a list [E_x, E_y, E_z] of three␣
↪components

        Returns:
        E: the uniform electric field, a list [E_x, E_y, E_z] of three␣
↪components
        '''
```

```
        return E


    def radial_E_field(self, r, V, center = [0,0,0]):
        '''
        This is a particular configuration of the electric field.
        E = V * (r - center) is the formula used

        Arguments:
        self
        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

        Returns:
        The electric field [E_x, E_y, E_z], a list with 3 components
        '''



        #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 * np.array(dr)
        return E

    def E_field_from_expression(self):
        '''
        As fields are generated numerically as of now, this method is not used.
        This method would be used to generate the Electric field using a␣
↪symbolic expression.
        '''



        E_x, E_y, E_z = sm.symbols("E_x E_y E_z")
        # do something

    def B_field__from_expression(self):
        '''
        As fields are generated numerically as of now, this method is not used.
        This method would be used to generate the Magnetic field using a␣
↪symbolic expression.
        '''
```

```
        B_x, B_y, B_z = sm.symbols("B_x B_y B_z")
        # do something

    def uniform_B_field(self, B):
        '''
        This is a particular configuration of the magnetic field,
        a uniform magnetic field

        Arguments:
        self
        B: the uniform magnetic field, a list [B_x, B_y, B_z] of three␣
↪components

        Returns:
        B: the uniform magentic field, a list [B_x, B_y, B_z] of three␣
↪components
        '''

        return B

    def helmholtz_coil_B_field(self, n, I, R, B_hat, mu_0):
        '''
        This is a particular configuration of the magnetic field,
        one that is generated by a Helmholtz coil

        Arguments:
        self
        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:
        the magnetic field [B_x, B_y, B_z], a list of 3 components
        '''


        return ( \
                (4/5)**1.5 * ( (mu_0 * n * I) / (R) ) * np.array(B_hat) \
               )

    def two_helmholtz_B_field(self, n1, I1, R1, B1_hat, n2, I2, R2, B2_hat,␣
↪mu_0):
        '''
        This is a particular configuration of the magnetic field,
```

```
        one that uses two Helmholtz coils at an angle

        Arguments:
        self
        n1: number of turns in the coil 1
        I1: current in the coil 1
        R1: radius of the coil 1
        B1_hat: the direction of the magnetic field generated by coil 1, i.e␣
↪the axis of the coil 1
        n2: number of turns in the coil 2
        I2: current in the coil 2
        R2: radius of the coil 2
        B2_hat: the direction of the magnetic field generated by coil 2, i.e␣
↪the axis of the coil 2
        mu_0: the constant mu_0 to be passed in using the constants in the␣
↪constants.ipynb file

        Returns:
        the resulant magnetic field of the two Heloltz coils [B_x, B_y, B_z], a␣
↪list of 3 components
        '''

        B1 = helmholtz(self, n1, I1, R1, mu_0, B1_hat)
        B2 = helmholtz(self, n2, I2, R2, mu_0, B2_hat)

        #Calculate the resultant of two magnetic fields
        B_hat = B1_hat + B2_hat
        return B_hat

    def magnteic_mirror_like_B_field(self):
        '''
        A magnetic field configuration such that grad B is parallel to B.
        '''
        pass
```

[ ]:

```
```

[ ]:

```
#Other notebooks may be imported here if required.
```