batch

April 9, 2022

This file instantiates objects of the class Particle in the particle.ipynb file, using initial velocities and initial positions generated in the sampling.ipynb file.

[1]:
```
import numpy as np # For computations
import csv # To read csv file of position and velocities required to initialize␣
↪particles
import import_ipynb
from particle import Particle
from field import Field
```

```
importing Jupyter notebook from particle.ipynb
importing Jupyter notebook from field.ipynb
```

[ ]:
```
# Probably good idea to make plots, write to csv and other things in another␣
↪notebook. Let's see.
```

[1]:
```
class Batch:
    '''
    An instance object of the step class can:
    1. initialize particles with positions and velocities passed from within␣
↪the setup or
    to be read from saved files
    2. update each particle under the action of magnetic and electric fields

    SUGGESTIONS FOR IMPROVEMENT
    Could be extended to:
    1. Save particle positions and velocities after a period of simulation
    '''

    def __init__(self, file_with_filenames = \
                 '/home/kushik/Kushik/VIT/Eighth semester/MagneticMirror/
↪csvfiles/sampling/available files.csv', \
                 read_directory = '/home/kushik/Kushik/VIT/Eighth semester/
↪MagneticMirror/csvfiles/sampling/', \
                 save_directory = '/home/kushik/Kushik/VIT/Eighth semester/
↪MagneticMirror/csvfiles/saved/'):
        self.file_with_filenames = file_with_filenames
```

```
        '''available files.csv contains the files from which positions and␣
↪velocities
        could be read
        Optionally they could just be generated during the program using␣
↪sampling methods
        in the Sampling class in sampling.ipynb
        However for better performance, it might be better to sample positions␣
↪and velocities
        and store them in csv files
        and later read them from the files and use them.
        Sampled positions and velocities can also be reused in this way.
        '''
        self.read_directory = read_directory # The location of the file with␣
↪path self.file_with_filenames

        self.filenames, self.r_filenames, self.v_filenames = self.
↪read_available_filenames()
        '''self.filenames would then be a list of availble files to read from
           self.r_filenames would be the list of available files with position␣
↪written on them
           self.v_filenames would be the list of available files with␣
↪velocities written on them'''

        self.particles = []
        # The list of the particles created during an instance of a Batch␣
↪object.
        # Or one static state of number of particles, in a dynamic running of␣
↪plasma

        self.number_of_files_available = len(self.filenames)
        self.filenames_descriptions = dict(zip(self.filenames,['' for i in␣
↪range(self.number_of_files_available)]))
        #Holds descriptions of files stored in the list filenames
        #like Temperature for the Maxwellian sampling methods

        self.save_directory = save_directory
        '''
        Files with states of particles are saved in this directory.
        '''

    def __string__(self):
        '''
        As string for an instance of the Step class has not been defined.
        However a paramter like the starting time of this step, number of␣
↪particles in this step, etc.
        could be printed for information
```

```
        '''
        pass

    ### FILE READING SECTION

    def read_available_filenames(self):
        '''
        Reads the names of the available files from available files.csv file
        from the directory (including the file path) self.file_with_filenames

        Arguments:
        self

        Returns:
        filenames: Names of the available files
        r_filenames: Names of the available files with positions of particles
        v_filenames: Names of the available files with velocities of particles
        '''


        rows = []
        filenames = []
        r_filenames = []
        v_filenames = []
        with open(self.file_with_filenames, 'r') as csvfile:
            # creating a csv reader object
            csvreader = csv.reader(csvfile)

            # extracting field names through first row
            #fields = next(csvreader)

            # extracting each data row one by one
            for row in csvreader:
                rows.append(row)
        for row in rows:
            empty = ''
            for char in row:
                empty += char
            filenames.append(empty)
        for filename in filenames:
            newfilename = filename.replace(self.read_directory, '')
            r_or_v = newfilename.split()[1]
            if r_or_v == 'r':
                r_filenames.append(filename)
            else:
                if r_or_v == 'v':
                    v_filenames.append(filename)
```

```
                else:
                    raise Exception('There might be some error in filename or␣
↪how filename is processed in file reading functions')

        return filenames, r_filenames, v_filenames

    def print_available_files(self):
        '''
        Prints the names of the available files from self.filenames

        Arguments:
        self

        Returns:
        nothing
        '''


        print(self.filenames)

    def print_available_r_files(self):
        '''
        Prints the names of the available files with positions from self.
↪r_filenames

        Arguments:
        self

        Returns:
        nothing
        '''


        print(self.r_filenames)

    def print_available_v_filenames(self):
        '''
        Prints the names of the available files with velocities from self.
↪v_filenames

        Arguments:
        self

        Returns:
        nothing
        '''
```

```
        print(self.v_filenames)

    def read_r_or_v_file(self, index):
        '''
        Reads a file from self.filnames of available files to read positions or␣
↪velocities from
        print_available_files can be called to check the available files and␣
↪determine the index to be passed in

        Arguments:
        self
        index: the index on the self.filenames list of which the corresponding␣
↪file is to be read.

        Returns:
        a list of the rows of the file read
        '''


        rows = []
        with open(self.filenames[index], 'r') as csvfile:
            csvreader = csv.reader(csvfile)

            for row in csvreader:
                rows.append(row)
        return rows

    def read_r_file(self, index):
        '''
        Reads a file from self.r_filnames of available files to read positions␣
↪from
        print_available_r_files can be called to check the available files and␣
↪determine the index to be passed in

        Arguments:
        self
        index: the index on the self.r_filenames list of which the␣
↪corresponding file is to be read.

        Returns:
        a list of the rows of the file read
        '''


        rows = []
```

```
        with open(self.r_filenames[index], 'r') as csvfile:
            csvreader = csv.reader(csvfile)

            for row in csvreader:
                rows.append(row)
        return rows

    def read_v_file(self, index):
        '''
        Reads a file from self.r_filnames of available files to read velocities␣
↪from
        print_available_v_files can be called to check the available files and␣
↪determine the index to be passed in

        Arguments:
        self
        index: the index on the self.v_filenames list of which the␣
↪corresponding file is to be read.

        Returns:
        a list of the rows of the file read
        '''


        rows = []
        with open(self.v_filenames[index], 'r') as csvfile:
            csvreader = csv.reader(csvfile)

            for row in csvreader:
                rows.append(row)
        return rows

    def reshaper(self, array_from_file, n):
        '''
        Reshapes a list for example, the list returned by read_r_or_v_file,␣
↪read_r_file or read_v_file
        into lists of positions or velocities of particles (3 components)

        Arguments:
        self
        array_from_file: array to be reshaped,
        for example the list returned by read_r_or_v_file, read_r_file or␣
↪read_v_file
        n: Number of particles represented in the array

        Returns:
```

```
        An array reshaped into a list of position or velocity - like arrays␣
↪(with 3 components)
        '''


        reshaped_array = np.array(array_from_file).reshape((n, 3))
        return reshaped_array

    def read_r_or_v_file_and_reshape(self, index, N, n):
        '''
        Reads file on the index: index, of seld.filenames
        and returns a list of position or velocity -like shaped arrays

        Arguments:
        self
        index: the index on the self.filenames list of which the corresponding␣
↪file is to be read.
        N: Number of particles represented whose positions or velocities are␣
↪saved in the file
        n: Number of particles to use in the simulation. n <= N is allowed

        Returns:
        An array reshaped into a list of position or velocity - like arrays␣
↪(with 3 components)
        '''

        if n > N:
            n = N
            #Default n for bad value

        array_from_file = self.read_r_or_v_file(index)
        reshaped_array = self.reshaper(array_from_file, N)
        return reshaped_array[:n]

    def read_r_file_and_reshape(self, index, N, n):
        '''
        Reads file on the index: index, of seld.r_filenames
        and returns a list of position -like shaped arrays

        Arguments:
        self
        index: the index on the self.r_filenames list of which the␣
↪corresponding file is to be read.
        N: Number of particles represented whose positions are saved in the file
        n: Number of particles to use in the simulation. n <= N is allowed

        Returns:
```

```
        An array reshaped into a list of position - like arrays (with 3␣
↪components)
        '''

        if n > N:
            n = N
            #Default n for bad value

        array_from_file = self.read_r_file(index)
        reshaped_array = self.reshaper(array_from_file, N)
        return reshaped_array[:n]

    def read_v_file_and_reshape(self, index, N, n):
        '''
        Reads file on the index: index, of seld.v_filenames
        and returns a list of position -like shaped arrays

        Arguments:
        self
        index: the index on the self.v_filenames list of which the␣
↪corresponding file is to be read.
        N: Number of particles represented whose positions are saved in the file
        n: Number of particles to use in the simulation. n <= N is allowed
        Returns:
        An array reshaped into a list of velocity - like arrays (with 3␣
↪components)
        '''

        if n > N:
            n = N
            #Default n for bad value

        array_from_file = self.read_v_file(index)
        reshaped_array = self.reshaper(array_from_file, N)
        return reshaped_array[:n]

    def add_description_to_available_files(self, index, description):
        '''
        Adds description in self.filenames_description corresponding to
        a file at the index: index in self.filenames

        Arguments:
        self
        index: the index corresponding to the file in self.filenames whose␣
↪description is to be changed/added
        description: the description to be added, a string.
```

```
        Returns:
        nothing
        '''


        self.filenames_descriptions[ self.filenames[index] ] = description

    ### PARTICELS INTIALIZATION SECTION
    def initialize_particles_of_different_species(self, names, q_s, m_s, r0_s,␣
↪v0_s, n):
        #Currently initial acceleration is not taken as input

        '''
        Instantiates objects of the Particle class from particle.ipynb and adds␣
↪them to the list self.particles
        Arrays for position and velocities may either be obtained by reading␣
↪from saved files
        using reading methods in the current Step class
        or generated otherwise during the program

        Arguments:
        self
        names, q_s, m_s, r0_s, v0_s are arrays of name, q, m, r_0, v_0
        respectively for each particle arranged such that for all of these␣
↪arrays
        the i_th index represents q, m, r_0, v_0, a_0 of the i_th particle
        n: number of particles initialized, or the length of each of these␣
↪arrays which is the same

        Returns:
        nothing
        '''


        for i in range(n):
            self.particles.append(Particle(names[i], q_s[i], m_s[i], r0_s[i],␣
↪v0_s[i] ))
        ### Might need to adjust for a_0_s[i]

    def initialize_particles_of_same_species(self, name, q, m, r0_s, v0_s, n):
        '''
        Instantiates objects of the Particle class from particle.ipynb and adds␣
↪them to the list self.particles
        Arrays for position and velocities may either be obtained by reading␣
↪from saved files
        using reading methods in the current Step class
```

```
        or generated otherwise during the program

        Particles of the same species are created (same name, charge, mass)
        Arguments:
        self
        r0_s, v0_s, a_0_s are arrays of r_0, v_0. name, q and m are the values␣
↪for the species
        respectively for each particle arranged such that for all of these␣
↪arrays
        the i_th index represents q, m, r_0, v_0, a_0 of the i_th particle
        n: number of particles initialized, or the length of each of these␣
↪arrays which is the same

        Returns:
        nothing
        '''


        for i in range(n):
            self.particles.append(Particle(name, q, m, r0_s[i], v0_s[i]))

    ### FIELDS INITIALIZATION SECTION
    def initialize_fields(self):
        '''
        Instantiates an object of the Field class from field.ipynb and assigns␣
↪it to self.fields

        Arguments:
        self

        Returns:
        nothing
        '''


        self.fields = Field()

    ### TIME STEP SECTION
    def update_particles(self, dt, argsE, argsB, track):
        '''
        Updates each particle in self.particles

        Arguments:
        self
        dt: time step of update
        argsE: arguments required for generating the electric field at each␣
↪particle's position
```

```
        required to move the particles, by calling the method get_E_field which␣
↪is passed
        to the update method of each particle object in self.particles
        argsB: arguments required for generating the magnetic field at each␣
↪particle's position
        required to move the particles, by calling the method get_B_field which␣
↪is passed
        to the update method of each particle object in self.particles
        track: list of particles in this batch that is to be updated to be␣
↪tracked
        For example: [1,30,18] means
        track the 1st 30th and 18th particles in this batch

        Returns:
        position_and_velocity: A dictionary whose keys are the particles to be␣
↪tracked in this batch
        i.e, the elements of track (input argument to this function), and␣
↪values are
        tuples of (position, velocity) after the update
        '''

        self.initialize_fields()

        position_and_velocity = dict()

        args = (self.fields, dt, argsE, argsB)

        for particle in self.particles:
            particle.update(args)
            current_index = self.particles.index(particle)
            if current_index in track:
                position_and_velocity[current_index] = (particle.r, particle.v)
            # Currently position and velocity after the update is tracked

        return position_and_velocity

    ### REMOVE PARTICLES SECTION
    def remove_particles(self):
        pass

    def absorb_particles(self):
        pass

    ### WRITE TO FILE
    def write_to_csv_file(self, particles, n, details = ''):
        '''
```

```
        Writes particle details to csv file.

        Arguments:
        self
        particles: a list of particles which is itself a list [name, mass,␣
↪charge, r, v] for a particle
        n: number of particles to be saved
        details: a string describing some details about the particles
        for example for the sample_Maxwellian_velocity_all_random_direction␣
↪method, the temperature used,
        time of running etc.

        Returns:
        nothing

        sample file name: ###YET TO DECIDE
        '''


        filename = ''
        filename += str(n)
        filename += ' '
        filename += details
        filename += ' '

        name = self.save_directory + filename

        particles.tofile(name, sep = ',', format = '%s')
                            ### WARNING: Data is written in string format so␣
↪needs to be converted to float
                            ### for charges, masses, r_s, v_s when reading␣
↪(relevant in run.ipynb)
        self.write_file_name(self, name)

    def write_file_name(self, added_file):
        '''
        Writes the name of the file to which an array of particles is written␣
↪to,
        in the available files.csv file

        Arguments:
        self
        added_file: the file to which an array of particles is written to

        Returns:
        nothing
        '''
```

```
        name = self.save_directory + 'available files.csv'
        with open(name, 'a') as f_object:
            writer_object = csv.writer(f_object)
            writer_object.writerow(added_file)
            f_object.close()
```

[ ]: