sampling

April 9, 2022

This file samples initial velocities and initial positions required to instantiate objects of the Particle class (from particle.ipynb) in step.ipynb file.

[3]:
```
import numpy as np # For computations
import scipy.stats as stats # For probability distribution function
import math
import csv # To write positions and velocities to csv files

from datetime import datetime # To get date and time of writing the csv files
#If required import other notebooks at the end of this notebook
```

[4]:
```
# assert dimensions in all position and velocity samplings
# always return numpy array in the specific sampling strategy functions so that␣
↪writing numpy array to csv works
class Sampler:
    '''
    An instance object of the Sampler class can:
    1. sample positions and velocities of particles based on different␣
↪strategies which could be used directly
    in a program
    2. write the sampled positions and velocities to csv files which can later␣
↪be read

    SUGGESTIONS FOR IMPROVEMENTS
    1.
    '''


    def __init__(self, directory = '/home/kushik/Kushik/VIT/Eighth semester/
↪MagneticMirror/csvfiles/sampling/'):

        self.directory = directory

        # The argument directory is the folder means where you want to save the␣
↪csv files and read them from

    def __string__(self):
```

```
        '''
        Currently no string is defined for an object instance of the Sampler␣
↪class
        '''
        pass

    def uniform_random_unit_vector(self):
        '''
        Generates a uniformly distributed random unit vector in 3 dimensions
        that may be used as direction for velocity or position relative to
        the center of the coordinate system.

        Arguments:
        nothing

        Returns:
        a uniformly distributed random unit vector in 3 dimensions as a list␣
↪[x, y, z]
        '''
        phi = np.random.uniform(0,np.pi*2)
        costheta = np.random.uniform(-1,1)

        theta = np.arccos(costheta)
        x = np.sin(theta) * np.cos(phi)
        y = np.sin(theta) * np.sin(phi)
        z = np.cos(theta)
        return np.array([x,y,z])

    def sample(self, args_r, args_v, n):
        '''
        Currently positions and velocities are sampled independently in␣
↪sample_position and sample_velocity
        methods
        '''
        pass


    def sample_position(self, args_r, n):
        '''
        Samples the position of particles using a position sampling strategy.
        Currently sample_same_given_distance_all_random_direction method is␣
↪used.

        Arguments:
        self
        args_r: arguments required to call the currently used position sampling␣
↪method,
```

```
        sample_same_given_distance_all_random_direction
        n: Number of particles for which the sampling is to be done

        Returns:
        An array of sampled positions for n particles according to the␣
↪currently used position sampling strategy.
        '''


        args_r = d
        positions = sample_same_given_distance_all_random_direction(self, d, n)
        return positions

    def sample_velocity(self, args_v, n):
        '''
        Samples the velocity of particles using a velocity sampling strategy.
        Currently sample_Maxwellian_velocity_all_random_direction strategy is␣
↪used.

        Arguments:
        self
        args_v: arguments required to call the currently used velocity sampling␣
↪method,
        sample_Maxwellian_velocity_all_random_direction
        n: Number of particles for which the sampling is to be done

        Returns:
        An array of sampled velocities for n particles accroding to the␣
↪currently used position sampling strategy.
        '''


        args_v = v_median, K, T, m
        velocities = self.
↪sample_Maxwellian_velocity_all_random_direction(v_median, K, T, m, n)
        return velocities

    def sample_same_given_position(self, r, n):
        '''
        All particles are sampled in the same position, for example if they all␣
↪enter through a valve.

        Arguments:
        self
        r: the position of all the partilces to be sampled, which is the same␣
↪for all
```

```python
        n: the number of particles to be sampled

        Returns:
        An array of sampled positions for n particles such that the position␣
↪for each particle is the same
        and equal to the input agrument r
        '''


        positions = []
        for i in range(n):
            positions.append(r)

        return np.array(positions)

    def sample_same_given_distance_all_random_direction(self, d, n):
        '''
        All particles are sampled at the same distance the center of the␣
↪coordinate system
        for example if all particles begin 1 meter away from an electrode in␣
↪any direction, inside a box trap

        Arguments:
        self
        d: the distance of all the particles to be sampled, which is the same␣
↪for all
        n: the number of particles to be sampled

        Returns:
        An array of sampled positions for n particles such that each particle␣
↪is distance d away from the origin
        of the coordinate system
        '''


        positions = []

        for i in range(n):
            positions.append(d * self.uniform_random_unit_vector())

        return np.array(positions)

    def sample_same_given_velocity_same_direction(self, v, n):
        '''
        All particles are sampled to such thate they have the same velocity,
        for example if they are all pushed by a pump in the same direction
```

```
        Arguments:
        self
        v: the velocity of the particles to be sampled, which is the same for␣
↪all particles
        n: the number of particles to be sampled

        Returns:
        An array of sampled velocities for n particles such that each particle␣
↪has velocity equal
        to the input argument v
        '''


        velocities = []
        for i in range(n):
            velocities.append(v)

        return np.array(velocities)

    def sample_same_given_speed_all_random_direction(self, s, n):
        '''
        All particles are sampled such that they have the same speed
        but random directions of motion

        Arguments:
        self
        s: the speed of the particles to be sampled, which is the same for all␣
↪the particles
        n: the number of particles to be sampled

        Returns:
        An array of sampled velocities for n particles such that each particle␣
↪has speed equal
        to the input argument s
        '''


        velocities = []

        for i in range(n):
            velocities.append(s * self.uniform_random_unit_vector())

        return np.array(velocities)

    def sample_velocity_uniformKE_same_given_direction(self):
        '''
```

```
        All the particles have the same kinetic energy, for example if they␣
↪have been accelerated
        through the same potential.
        They would have velocities depending on their masses.
        '''
        pass

    def sample_velocity_uniformKE_all_random_directions(self, n):
        pass

    def sample_Maxwellian_speed(self, v_median, K, T, m, n):
        '''
        This method samples Maxwellian speeds (ONLY) for particles and␣
↪therefore is used
        in other methods that also generate directions, for sampling of␣
↪velocities

        Arguments:
        self
        v_median: The median velocity of the distribution
        K: The Boltzmann constant, to be passed from the constants.ipynb file
        T: The thermodynamic temperature of the particle distribution to be␣
↪sampled
        m: the mass of the particles to be sampled
        For simplicity this method currently only samples speeds for particles␣
↪of the same mass, at a time
        n: the number of particles to be sampled

        Returns:
        An array of Maxwellian distributed speeds dependent on the arguments␣
↪v_meidan, K, T and m for n particles
        '''



        alpha = math.sqrt(K * T / m)
        speeds = stats.maxwell.rvs(loc = v_median, scale = alpha, size = n)
        return speeds

    def sample_Maxwellian_velocity_same_given_direction(self, v_median, K, T,␣
↪m, v_hat, n):
        '''
        Particles are sampled such that they have Maxwellian distributed speeds␣
↪but
        are all moving in the same specified direction, for example if they are␣
↪moving in a ion trap system
```

```
        Arguments:
        self
        v_median: The median velocity of the distribution
        K: The Boltzmann constant, to be passed from the constants.ipynb file
        T: The thermodynamic temperature of the particle distribution to be␣
↪sampled
        m: the mass of the particles to be sampled
        For simplicity this method currently only samples velocities for␣
↪particles of the same mass, at a time
        v_hat: the direction of motion which is the same for all the particles
        n: the number of particles to be sampled

        Returns:
        An array of Maxwellian distributed speeds dependent on the arguments␣
↪v_median, K, T and m for n particles
        and direction of motion equal to the agrument v_hat for all particles
        '''


        speeds = self.sample_Maxwellian_speed(v_median, K, T, m, n)
        velocities = np.outer(speeds, v_hat)

        assert len(velocities) == n, 'Dimensions don\'t match. There is some␣
↪error in multiplying \
        speeds and the direction'

        return np.array(velocities)

    def sample_Maxwellian_velocity_same_random_direction(self, v_median, K, T,␣
↪m, n):
        '''
        Particles are sampled such that they have Maxwellian distributed speeds␣
↪but
        are all moving in the same random direction

        Arguments:
        self
        v_median: The median velocity of the distribution
        K: The Boltzmann constant, to be passed from the constants.ipynb file
        T: The thermodynamic temperature of the particle distribution to be␣
↪sampled
        m: the mass of the particles to be sampled
        For simplicity this method currently only samples velocities for␣
↪particles of the same mass, at a time
        n: the number of particles to be sampled
```

```
        Returns:
        An array of Maxwellian distributed speeds dependent on the arguments␣
↪v_median, K, T and m for n particles
        moving in the same random direction
        '''


        speeds = self.sample_Maxwellian_speed(v_median, K, T, m, n)
        direction = self.uniform_random_unit_vector()
        velocities = np.outer(speeds, direction)

        assert len(velocities) == n, 'Dimensions don\'t match. There is some␣
↪error in multiplying \
        speeds and the direction'

        return np.array(velocities)

    def sample_Maxwellian_velocity_all_random_direction(self, v_median, K, T,␣
↪m, n):
        '''
        Particles are sampled such that they have Maxwellian distributed speeds
        but are moving in uniformly distributed random directions

        Arguments:
        self
        v_median: The median velocity of the distribution
        K: The Boltzmann constant, to be passed from the constants.ipynb file
        T: The thermodynamic temperature of the particle distribution to be␣
↪sampled
        m: the mass of the particles to be sampled
        For simplicity this method currently only samples velocities for␣
↪particles of the same mass, at a time
        n: the number of particles to be sampled dependent on the arguments␣
↪v_median, K, T and m for n particles
        moving in uniformly distributed random directions

        Returns:
        An array of Maxwellian distributed speeds for n particles moving in
        uniformly distributed random directions
        '''


        speeds = self.sample_Maxwellian_speed(v_median, K, T, m, n)
        velocities = []
        for i in range(n):
```

```
            velocities.append(speeds[i] * np.array(self.
↪uniform_random_unit_vector()))

        assert len(velocities) == n, 'Dimensions don\'t match. There is some␣
↪error in multiplying \
        speeds and directions'

        return np.array(velocities)

    def sample_parabolic_speed_f1(self, n, loc, scale):
        '''
        First 1-x^2 distributed speeds are generated based on uniform randomly␣
↪sampled x from [0, 1)
        The speeds are scaled by the variance and then shifted by the mean.

        Arguments:
        self
        n: the number of particles to be sampled
        loc: the mean speed of the particles, passed to scipy rdist
        scale: the shape of the distribution, passed to scipy rdist

        Returns:
        An array of Parabolic distributed speeds
        '''

        speeds = stats.rdist.rvs(c=4, loc=loc, scale=scale, size=n)
        return np.array(speeds)

    def sample_parabolic_speed_f1_all_random_direction(self, n, loc, scale):
        '''
        The speeds generated by sample_parabolic_speed_f1 are multiplied
        by uniform randomly generated velocity directions.

        Arguments:
        self
        n: the number of particles to be sampled
        loc: the mean speed of the particles, passed to scipy rdist
        scale: the shape of the distribution, passed to scipy rdist

        Returns:
        An array of Parabolic distributed speeds with vecloity directions␣
↪uniform randomly distributed
        '''


        speeds = self.sample_parabolic_speed_f1(n, loc, scale)
        velocities = []
```

```
        for i in range(n):
            velocities.append(speeds[i] * np.array(self.
↪uniform_random_unit_vector()))

        assert len(velocities) == n, 'Dimensions don\'t match. There is some␣
↪error in multiplying \
        speeds and directions'

        return np.array(velocities)

    def write_to_csv_file(self, arr, r_or_v, strategy, n, details = ''):
        '''
        Writes sampled positions or velocities to csv files.

        Arguments:
        self
        arr: array of positions or velocities of sampled particles
        r_or_v = 'r' meaning that the array arr contains sampled positions
        or 'v' meaning that the array arr contains sampled velocities
        others are not accepted
        strategy: name of the method used to generate this array,
        like 'sample_Maxwellian_velocity_all_random_directions'
        n: number of particles for which the sampling was done
        details: a string describing some details about the sampling,
        for example for the sample_Maxwellian_velocity_all_random_direction␣
↪method, the temperature used

        Returns:
        nothing

        sample file name: '10 v sample_Maxwellian_velocity_all_random_direction␣
↪24-01-2022 14:46:53:935235'
        this means for 10 particles velocities have been written at date␣
↪24-01-2022, at time 14:46:53:935235
        -- without the details
        the details go after the strategy
        '''


        filename = ''
        filename += str(n)
        filename += ' '
        if r_or_v == 'r' or r_or_v == 'v':
            filename += r_or_v
        else:
            raise Exception('Invalid value for r_or_v, only "r" or "v" allowed.
↪')
```

```
        filename += ' '
        filename += strategy
        filename += ' '

        if details != '':
            filename += details
            filename += ' '

        # datetime object containing current date and time
        now = datetime.now()
        # dd/mm/YY H:M:S
        dt_string = now.strftime("%d-%m-%Y %H:%M:%S:%f")

        filename += dt_string
        name = self.directory + filename

        arr.tofile(name, sep = ',', format = '%f')
        print("SUCCESS")
        print(name)
        self.write_file_name(name)

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

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

        Returns:
        nothing
        '''


        name = self.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()
```

```
[ ]:
```