**Overview**

This program will simulate a behaviour of an N-body system interacting through the Lennard-Jones pair potential at user-defined conditions (temperature, density). Particle positions and velocities will be updated using a velocity Verlet time integration method. Applying periodic boundary condition method and minimum image convention the trajectories of particles will be simulated using Visual Molecular Dynamics (VMD) program. The program will generate the output file containing the radial distribution function and total energy as a function of time.

This document describes the classes and methods written for this program, how they interface to each other and outlines any algorithms implemented. All code is written in python.

**Class structure**

The following outlines the structure of the program:

**Particle3D Class**

Each instance represents a particle’s properties (label, mass, position, velocity) in 3-dimensional space.

**Properties**

|  |  |  |
| --- | --- | --- |
| Name | Type | Notes |
| label | str | ‘Name’ of the particle |
| position | np.array([x\_pos, y\_pos, z\_pos]) | Position of the particle in 3-dimensional space |
| velocity | np.array([vel\_x, vel\_y, vel\_z]) | Velocity of the particle in 3-dimensional space |
| mass | float | Mass of particle |

**Constructor**

|  |  |
| --- | --- |
| Arguments | Notes |
| str label, np.array([x\_pos, y\_pos, z\_pos]), np.array([vel\_x, vel\_y, vel\_z]), float m | Creates a particle at position (x\_pos, y\_pos, z\_pos) with a velocity vector (vel\_x, vel\_y, vel\_z), mass m and name ‘label’. |

**Methods**

**\_\_str\_\_()**

Returns a string containing a particle’s name and position in the following format:

*<label> <x\_pos> <y\_pos> <z\_pos>*

**kinetic\_energy()**

Returns the kinetic energy of the particle based on the following equation:

where Ek represents kinetic energy, m represents the reduced particle mass and **v** is the particle’s velocity in 3-dimensional space.

**potential\_energy()**

Returns the potential energy of an interaction between two Particle3D instances using the following equation:

where U(**r**) represents potential energy, and **r** represents the separation vector between two particles.

**leap\_velocity(dt, force)**

Updates the velocity of the particle for a given force and time step, as represented by the following equation:

where and represents the particle’s updated and initial velocity respectively, dt represents the time step and represents the pairwise force interaction.

**leap\_pos2nd(dt, force)**

Updates the particle’s position for a given time step at 2nd order, as represented by the following equation:

where and represent the particle’s updated and initial position respectively.

**Static Methods**

**parameter\_reader(file\_handle)**

Generated a Particle3D instance from file input, with the provided file input containing information about the particle in the following format:

*<x-pos> <y-pos> <z-pos> <vel\_x> <vel\_y> <vel\_z> <mass>*

**vector\_split(p\_1, p\_2)**

Returns the separation vector between two particles, ie:

where , are the position vectors of the particles.

**pair\_force(r\_cut, p\_1, p\_2)**

Returns the pairwise force for two particles p\_1 and p\_2, following the equation:

where is a separation vector between two particles.

A user-defined cutoff radius r\_cut is taken into account such that the returned output is 0 if the vector separation between two particles exceed r\_cut.

**MDUtilities Class**

This class contains static methods which set the initial positions of the particles as well as their initial velocities.

This class does not contain any instance methods, constructors or properties.

**Static Methods**

**setInitialPositions(rho, particles):**

Arranges all the particles in a face-centred cubic (fcc) lattice with spacing between particles dependent on density (rho). This method also returns a string output which tells the user the following:

1. Whether the atoms fill an fcc lattice completely.
2. The number of atoms placed on the fcc lattice.
3. The dimensions of the ‘box’ containing the particles.

**setInitialVelocities(temp, particles):**

Sets the initial velocity of the particles at random, but rescaled to take into account temperature as defined by the user.

**ListMethods Class**

This class contains static methods which operate on a list of Particle3D instances.

This class does not contain any instance methods, constructors or properties.

**traj\_output(listObjects, outfile):**

Writes a list of Particle3D instances (listObjects) containing a user-defined number of particles to a file in the format defined in the class Particle3D method \_\_str\_\_(). A loop is implemented such that the method \_\_str\_\_() in class Particle3D keeps generating a string output for each particle in listObjects, with each string being subsequently written to the file. The loop continues until all the strings corresponding to each instance in listObjects have been written to the file.

**force\_sum(listObjects):**

Returns the sum of all pairwise force interactions in the list of Particle3D instances (listObjects) containing a user-defined number of particles. The sum is done by calculating individual pairwise forces using the method ‘pair\_force’ in class Particle3D and then taking a sum of all possible pairwise forces for the particles in listObjects.

**rdf\_histogram(listObjects):**

This method returns a histogram of all possible particle separation vectors for all Particle3D instances in listObjects through the use of the method ‘histogram’ in the numpy package.

**histogram\_norm(hist\_data):**

This method normalises the histogram data generated from the method ‘rdf\_histogram’.

**posUpdate(listObjects):**

Returns the updated list of particles (Particle3D objects) with changed postions (due to LJ interactions between particles).

**velUpdate(listObjects):**

Returns the updated list of particles (Particle3D objects) with changed velocities (due to LJ interactions between particles).

**E\_system(listObjects):**

Returns the kinetic and potential energy of the entire system.

**TE\_system():**

Returns the total energy of the entire system.

**PBC():**

Returns the list of the positions

**meansq\_displacement(listObjects, listPoints):**

Returns the mean squared displacement of all Particle3D instances in listObjects between their initial positions and their position at time t, as defined by the equation below:

where MSD represents squared displacement, representing particle positions at time t, representing initial particle position and represents the number of Particle3D instances contained in listObjects.

**‘Main\_Program’**

This class contains the main program which simulates the N-body system as described in the overview, with a user-defined set of initial conditions (no. of particles, density, temperature).

This method does not contain any constructors, properties or instance methods.

**Static Methods**

**main()**

The main method will require the user to input the following variables:

1. Number of particles
2. Density
3. Temperature
4. File input containing parameters of a single particle
5. Names of 2 file outputs containing the following:
   1. Trajectory of all particles
   2. Total energy as a function of time

The method then generates a number of Particle3D instances and stores these in a list. The methods ‘setInitialPositions’ and ‘setInitialVelocities’ set the particle positions such that they have an fcc arrangement whilst taking into account the user-defined density, and particle velocities are set at random but takes into account the temperature set by the user. Afterwards, the method ‘loop’ is executed to continuously update the particles’ positions and velocities, while taking into account periodic boundary conditions and minimum image convention. A trajectory output file is then generated and this is used to simulate the N-Body sytem through VMD. An output file is also generated containing the total energy as a function of time. The matplotlib package is then used to plot the radial distribution function, which is generated by the methods ‘rdf\_histogram’ and ‘histogram\_norm’ in (name tbd). Lastly, a string is outputted to the screen which contains information about the Mean Squared Displacement of the particles over time.

\*formatting for this needs a bit of work\*

**loop(listObjects, listPoints, outfile\_traj, outfile\_energy)**

This method updates the following for all Particle3D instances in listObjects: velocity, position, kinetic energy and potential energy over a defined range ‘numstep’. The following shall outline how this is done.

Firstly, a loop is created over a range ‘numstep’. The following parameters are then updated/evaluated:

1. Position and velocity

The method posUpade(listObjects) updates the position of all the particle instances by applying the leap\_pos2d(dt,force) method to each particle3D object from the list in a loop . The method velUpdate(listObjects) updates the velocity of all the particle instances by applying the leap\_velocity(dt, force) method to each particle particle3D object from the list in a loop

Once all the velocities and positions of all Particle3D instances in listObjects are updated, listObjects is appended to a new list called ‘listPoints’. Lastly, trajectory parameters are written to an output file by using the method ‘traj\_output’ in class ListMethods. The updates repeat over the range ‘numstep’ with trajectory parameters subsequently being written to the output file.

1. Total energy

The method E\_system() calculates the kinetic and potential energies of the entire system. Kinetic and potential energies of each molecule are calculated using kinetic\_energy() and potential\_energy() methods respectively and added up for all molecules in in a loop. The TE\_system() method adds up the kinetic and potential energies of the entire system.

**# Not to be included, just some extra notes:**

Just my understanding on pbc and min. image convention, hope this helps

**PBC**

Positions of all Particle3D instances in listObjects are updated

----begin PBC method-----

1. If position of particle in listObjects is within box dimensions, position remains the same
2. Else (ie. position of particle in listObjects is outside box dimension), send it to the other side of the box (algorithm to do this seems fairly simple)
3. Do this for all particles (ie. make a loop)

Right now what I’m thinking is that the implementation inside the ‘loop’ method (in the Main program) will just have these lines inserted inside the ‘for’ loop:

posUpdate(listObjects) # Update all particle positions

pbc(listObjects) # After update of position, enforce PBC

-----end PBC method----

**Min Image Convention**

With regards to minimum image convention, well I’ll be joining you in the lab. I have some ideas on it too, but hard to explain here.