Computer Modelling

Lennard-Jones System Design Document

PROJECT

Gabriel Laude (s1565983), Mikolaj Roguski (s1455244)

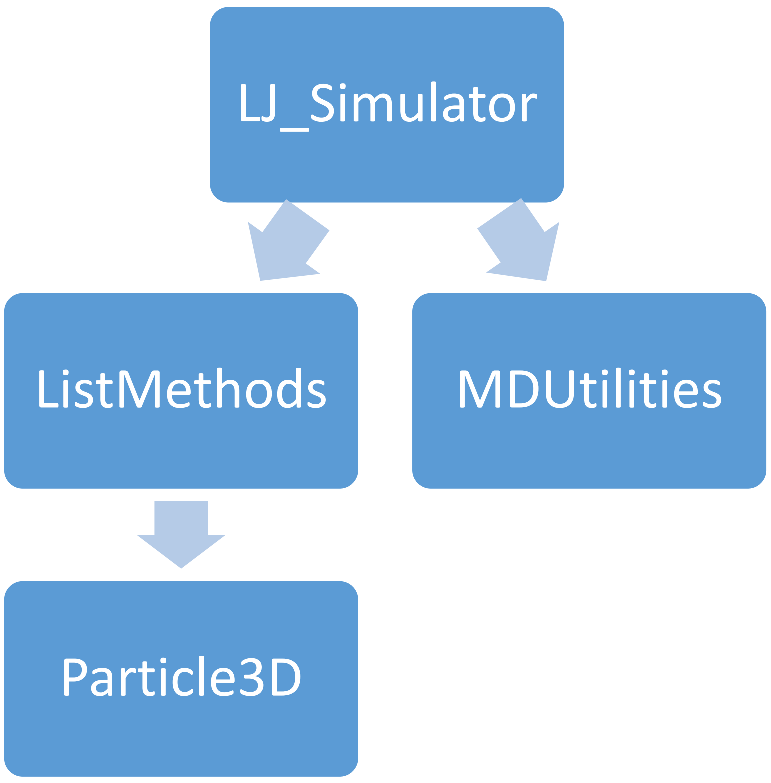
Overview

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 while keeping in mind Periodic Boundary Conditions (PBC) and minimum image convention. Output files containing particle trajectories, radial distribution and total energy as a function of time are generated. The trajectories of particles will be simulated using Visual Molecular Dynamics (VMD) program.

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 mass	Creates a particle at position (x_pos, y_pos, z_pos) with a velocity vector (vel_x, vel_y, vel_z), mass (defined here as a float value 'mass') 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:

$$E_k = \frac{1}{2} m \, |\boldsymbol{v}|^2$$

where E_k 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:

$$U(\boldsymbol{r}) = 4\left(\frac{1}{|\boldsymbol{r}|^{12}} - \frac{1}{|\boldsymbol{r}|^{6}}\right)$$

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:

$$\boldsymbol{v_{updated}} = \boldsymbol{v_{initial}} + \frac{dt \ \times \boldsymbol{F}(t)}{m}$$

where $\boldsymbol{v_{updated}}$ and $\boldsymbol{v_{initial}}$ represents the particle's updated and initial velocity respectively, dt represents the time step and $\boldsymbol{F}(t)$ 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:

$$\boldsymbol{r_{updated}} = \boldsymbol{r_{initial}} + (dt \times \boldsymbol{v}(t) + \left[dt^2 \times \left(\frac{\boldsymbol{F}(t)}{2m}\right)\right]$$

where $\boldsymbol{r_{updated}}$ and $\boldsymbol{r_{initial}}$ 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:

$$\boldsymbol{r}_{ij} = \boldsymbol{r}_i - \boldsymbol{r}_j$$

where $\boldsymbol{r}_i$, $\boldsymbol{r}_j$ 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:

$$F(\boldsymbol{r}) = 48\left(\frac{1}{|\boldsymbol{r}|^{14}} - \frac{1}{2|\boldsymbol{r}|^{8}}\right)\boldsymbol{r}$$

where $\boldsymbol{r}$ 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:

(i) Whether the atoms fill an fcc lattice completely.
(ii) The number of atoms placed on the fcc lattice.
(iii) 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(histData):

This method normalises the histogram data in the list histData (which contains the non-normalised radial distribution function).

posUpdate(listObjects):

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

velUpdate(listObjects):

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

E_system(listObjects):

Returns the kinetic and potential energy of the entire system.

TE_system(listEnergies):

Returns the total energy of the entire system and appends these to a list called 'listEnergies'.

pbc(listObjects, boxDim):

Given the dimensions of the box (boxDim), this method returns the list of updated position ensuring that all the particles that would exit the box enter the opposite wall of the box, ensuring minimum image convention.

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:

$$MSD(t) = \frac{1}{N}\sum_{i}|\boldsymbol{r}_i(t) - \boldsymbol{r}_{i0}|^2$$

where MSD represents squared displacement, $\boldsymbol{r}_i(t)$ representing particle positions at time t, $\boldsymbol{r}_{i0}$ representing initial particle position and N represents the number of Particle3D instances contained in listObjects.

LJ_Simulator

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:

(i) Number of particles
(ii) Cutoff radius
(iii) Density
(iv) Temperature
(v) File input containing parameters of a single particle
(vi) Names of 3 file outputs containing the following:
   a. Trajectory of all particles
   b. Total energy as a function of time
   c. Radial distribution function.

The method then generates a number of Particle3D instances defined by the user and stores these in a list. The methods setInitialPositions(rho, particles) and setInitialVelocities(temp, particles) 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' (see below) is executed to continuously update the particles' positions and velocities, while taking into account periodic boundary conditions and minimum image convention. Radial distribution function data generated by the method 'loop' is normalised by histrogram_norm(histData). A trajectory output file is then generated and this is used to visually simulate the N-Body sytem through VMD. Output files containing the normalised radial distribution function and total energy as a function of time are also generated. Lastly, a string is outputted to the screen which contains information about the Mean Squared Displacement of the particles over time.

loop(listObjects, listPoints, listEnergies, histData, outfile_traj, outfile_energy, outfile_rdf)

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 inside this loop:

(i) Position and velocity

The method posUpade(listObjects) updates the position of all the particle instances by applying the leap_pos2nd(dt,force) method to each Particle3D object from the list. It is followed by pbc(listObjects, boxDim) method to ensure that all the particles are kept in the box of given dimensions.

The method velUpdate(listObjects) updates the velocity of all the particle instances by applying the leap_velocity(dt, force) method to each Particle3D instance in listObjects.

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.

(ii) Total energy

The method E_system(listObjects) 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 adds this up for all molecules. The TE_system(listEnergies) method adds up the kinetic and potential energies of the entire system and then appends these to a list called 'listEnergies'. This is repeated for the entirety of the loop.

(iii) Radial distribution function

The method rdf_histogram(listObjects) is called upon and then a histogram of vector separations is generated and appended to list called 'histData'. This is then repeated for the entirety of the loop.