THE UNIVERSITY
of EDINBURGH

Design Document for Lennard-Jones potential N-body simulation

Marina Ruiz Sánchez-Oro

Cara Lynch

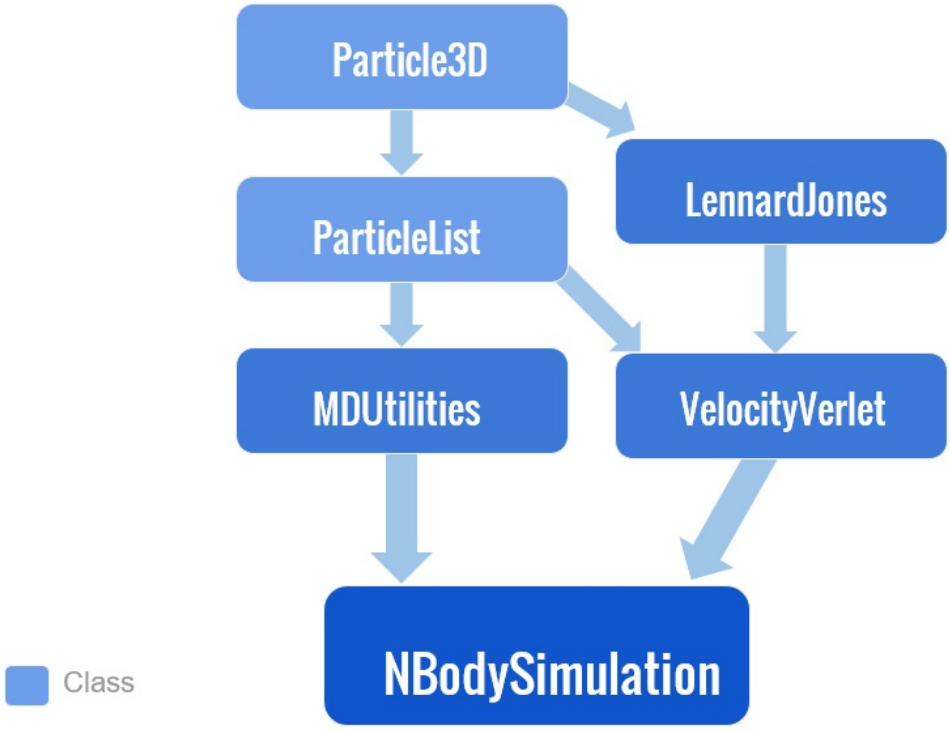
January 2017

Overview

This program will describe N-body systems interacting through the Lennard-Jones pair potential. It will be used to simulate, using periodic boundary conditions, a Lennard-Jones solid, fluid, and gas, and investigate equilibrium properties of the simulated systems. All quantities will be in reduced units.

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

Layout:



Note: Although we could have integrated the LennardJones and VelocityVerlet into our NBodySimulation code without creating separate files, we decided to separate them to be able to use the NBodySimulation for different potentials and time integration methods with little modification.

Particle 3D Class

Each instance represents a particle with 3D position (x, y, z), velocity (v_x, v_y, v_z) and mass, m.

Properties

Name	Type	Definition
`label`	**string**	Particle name
`position`	**Numpy array**	Particle position
`velocity`	**Numpy array**	Particle velocity
`mass`	**float**	Particle mass

Constructor

Arguments	Type	Comments
`label`	**string**	Creates a 3D particle with with a label defined by a position (`pos`),
`pos`	**Numpy array**	a velocity (`vel`) and a mass (`mass`)
`vel`	**Numpy array**	
`mass`	**float**	

Methods

`kineticEnergy(Particle)`

Returns the kinetic energy of a particle of mass m with speed v using the equation

$$E_{kinetic} = \frac{mv^2}{2} \tag{1}$$

`leapVelocity(Particle, Timestep, Force)`

Updates the velocity of a particle of mass m to the first order using the equation

$$\vec{v}(t + dt) = v(\vec{t}) + \frac{dt}{m} \cdot \vec{F} \tag{2}$$

`leapPos1st(Particle, Timestep)`

Updates the position of a particle to the first order using the equation

$$\vec{r}(t + dt) = r(\vec{t}) + dt \cdot \vec{F} \tag{3}$$

`leapPos2nd(Particle, Timestep, Force)`

Updates the position of a particle to the second order using the equation

$$r(t + \vec{dt}) = r(\vec{t}) + dt \cdot \vec{F} + \frac{dt^2}{2m} \cdot \vec{F} \tag{4}$$

`__str__(Particle)`

Prints the label and position coordinates of the particle.

Static Methods

`createparticle(File)`

Return a particle instance from a file entry containing the label, mass m, position (x, y, z) and velocity (v_x, v_y, v_z) of the particle.

`vectorseparation(Particle 1, Particle 2)`

Return the relative vector separation of two particles, i.e.

$$\vec{r_1} - \vec{r_2} = (x_1 - x_2)\hat{x} + (y_1 - y_2)\hat{y} + (z_1 - z_2)\hat{z} \tag{5}$$

Particle List Class

Each instance represents a list of N particles.

Properties

Name	Type	Definition
`labellist`	**list of stings**	Particle names
`positionlist`	**list of Numpy arrays**	Particle positions
`velocitylist`	**list Numpy arrays**	Particle velocities
`masslist`	**list of floats**	Particle mass

Constructor

Arguments	Type	Comments
`p1, p2, p3,...,pN`	**Particle3D instances**	Creates a list of N 3D particles (`pos`),

Methods

`totalKE(Particle List)`

Returns the total kinetic energy of a system of N particles using the `kineticEnergy` method described in the Particle3D class.

`listVelocityupdate(Particle List, Timestep, Force)`

Updates the velocity of each particle in the system of N particles to the first order using the `leapVelocity` method in the Particle3D class.

`listPos1st(Particle List, Timestep)`

Updates the position of each particle in the system to the first order using the `leapPos1st` method in the Particle3D class.

`listPos2nd(Particle List, Timestep, Force)`

Updates the position of each particle in the system to the second order using the `leapPos2nd` method in the Particle3D class.

`__str__(Particle List)`

Prints the label and position coordinates of the particles as a list.

Static Methods

`createlist(File)`

Returns a list of N particles from a file entry containing the label, mass m, position (x, y, z) and velocity (v_x, v_y, v_z) of each particle using the `createparticle` static method in the Particle3D class.

MDUtilities

Molecular Dynamics methods acting on ParticleList instances which sets the initial conditions for the simulation:

- box dimensions
- number of particles in each direction
- separation between particles
- particle positions
- initial velocities
- centre-of-mass motion
- Boltzmann factor

It organises the particles into a face-centered cubic (fcc) lattice and outputs the temperature of the system as well as the velocity of the centre-of-mass. Methods have not been detailed in this document as this code was provided with the project instructions.

LennardJones

Returns the force and potential energy between two particles interacting through the Lennard-Jones pair potential represented by Particle3D instances.

Methods

`PotentialEnergy(Particle 1, Particle 2, $r_c$)`

Returns the (reduced) potential energy of two particles with the equation ————————-

$$U(r) = 4\left[\frac{1}{r^{12}} - \frac{1}{r^6}\right] \tag{6}$$

where r_c is the cutoff radius above which the potential energy is set to 0 and the magnitude of the separation r is calculated using the Particle3D `vectorseparation` method.

`Force(Particle 1, Particle 2, $r_c$)`

Returns the force between two particles as a vector with the equation

$$\vec{F}(\vec{r}_{12}) = 48\left[\frac{1}{r^{14}} - \frac{1}{2r^8}\right]\vec{r}_{12} \tag{7}$$

where the vector separation $\vec{r}_{12}$ and its magnitude r are calculated using the Particle3D `vectorseparation` static method. r_c is the cutoff radius above which the force is set to 0.

VelocityVerlet

Performs time integration for a system of N particles' motion.

Method

`timeIntegration(ParticleList, force, dt, numstep)`

Updates the positions of a system of particles described by ParticleList using the velocity Verlet time integration algorithm. The particles experience a force (here calculated in the LennardJones algorithm) and the time integration is performed with timestep `dt` and a number of steps `numstep`. The methods described in the ParticleList class are used to update the particle position, force and velocity of the particles.

NBodySimulation

Simulates N-body systems interacting through the Lennard-Jones pair potential.

1. Reads the system parameters from an input file specifying particle properties as well as the density of the system, temperature, the radial distribution function, number and length of timesteps
2. Creates a list of Particle3D instances using the Particle3D and ParticleList classes
3. Sets the initial conditions of the simulation using MDUtilities
4. Computes the behaviour of the system over time using the velocity Verlet time integration algorithm
5. Outputs a trajectory file to produce the motion of the particles using VMD, determines the fluctuations of the total, potential and kinetic energy of the system, the average particle mean square displacements and the particle radial distribution

Methods

`PBC(Particle, boxSize)`

Algorithm that modifies the updated position of a particle to the correct value if it ends up outside the simulation box (whose volume is specified by `boxSize` in MDUtilities) boundaries with the Periodic Boundary Conditions approach when simulating a condensed phase. While the position vector of a particle is outside of the box (assumed cubic), the algorithm will update its position to return it to the inside of the box.

- If x, y, or $z < 0$ then the new x, y or z will be equal to $x + a$, $y + a$ or $z + a$
- If x, y, or $z > a$ then the new x, y or z will be equal to $x - a$, $y - a$ or $z - a$

Where a is the length of the sides of the box (assumed cubic).

`MinImage(Particle 1, Particle 2, boxSize)`

Algorithm that implements the minimum image convention when computing the separation between two particles. If the magnitude of the separation between two particles is bigger than half a box length away then an image will be closer, therefore the separation will be computed with relation to this image.

`MSD(ParticleList)`

Mean squared displacement algorithm that measures how the particles in a system deviate from

their initial positions using the equation

$$MSD(t) \equiv \frac{1}{N} \sum_{i=1}^{N} \mid \vec{r_i}(t) - \vec{r_{i0}} \mid^2 \tag{8}$$

`rdf(list of distances, ρ, N, r)`

Radial distribution function algorithm which calculates the probability to find a particle at a given distance from the reference particle. It provides information about the degree of ordering in the system. It is given by

$$RDF(r) \equiv \frac{1}{N_\rho} \langle \sum_{i,j} \delta(r_{ij} - r) \rangle \tag{9}$$

`particleDistances(ParticleList, Δt)`

Returns a list of all distances r_{ij} between pairs of particles (adhering to the minimum image convention) at regular time intervals Δt.

`normalisation(list of distances, ρ, r, dr)`

Bins the particle distances into a histogram of spacing `dr` and normalises the histogram data.