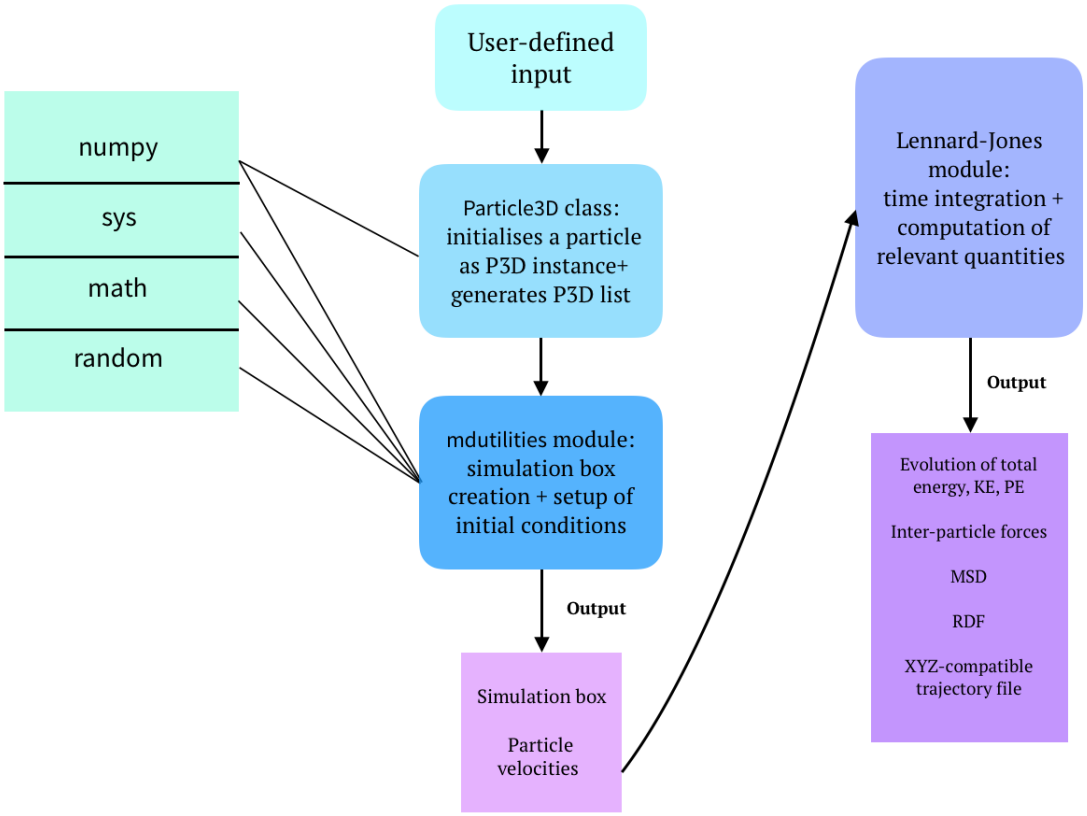
Design Document
Simulation of Argon States via Lennard-Jones Potential

1 Introduction

This program will simulate the solid, liquid, and gas phases of Argon where particles interact via the Lennard-Jones pair potential using the Velocity Verlet algorithm and periodic boundary conditions.



2 Scientific Background

The program uses the Lennard-Jones potential model for weakly-interacting systems, with reduced units. The following observables are computed:

- Total, potential, and kinetic energies of the system as a function of time
- Mean squared displacement as a function of time
- Radial distribution function

2.1 Lennard-Jones Interaction Potential

The potential energy of two particles at $\underline{r}_1$, $\underline{r}_2$ is given by

$$U(\underline{r}_1, \underline{r}_2) = 4\left[\frac{1}{r^{12}} - \frac{1}{r^6}\right] \tag{1}$$

Where $r = |\underline{r}_2 - \underline{r}_1|$.

2.2 Inter-Particle Force

The force on the particle at $\underline{r}_1$ is given by

$$\underline{F}_1 = -\underline{\nabla}_2 U(\underline{r}_1, \underline{r}_2) = 48 \left[\frac{1}{r^{14}} - \frac{1}{2r^8} \right] (\underline{r}_1 - \underline{r}_2) \tag{2}$$

From Newton's third law, $\underline{F}_2 = -\underline{F}_1$.

2.3 Mean Squared Displacement

The MSD measures how far particles have moved on average from their initial positions by a time t, and is defined as

$$\text{MSD}(t) = \langle |\underline{r}(t) - \underline{r}_0|^2 \rangle = \frac{1}{N} \sum_i |\underline{r}_i(t) - \underline{r}_{i0}|^2 \tag{3}$$

Where $\underline{r}_{i0}$ are the initial positions of the particles.

2.4 Radial Distribution Function

The RDF $g(r)$ is the probability of one particle being found at a given distance relative to another, averaged over time. It is given by

$$g(r) = \frac{1}{N \rho_0(r)} \left\langle \sum_{i,j} \delta(r_{ij} - r) \right\rangle \tag{4}$$

Where N is the number of particles in the system, $\rho_0(r)$ is the expected value of the RDF for a homogeneous system.

2.5 Units

To avoid floating point errors, the program uses reduced units, defined in relation to regular units as displayed in the table below.

Physical Quantity	**Reduced Unit**	**In Real Units**
Length	r	$\frac{r^*}{\sigma}$
Force	F	$\frac{F^* \sigma}{\epsilon}$
Energy	U	$\frac{U^*}{\epsilon}$
Density	ρ	$\rho^* \sigma^3$
Pressure	P	$\frac{P^* \sigma^3}{\epsilon}$
Temperature	T	$\frac{k_B T^*}{\epsilon}$
Time	t	$t^* \sqrt{\frac{\epsilon}{m\sigma^2}}$

Where * is used to denote the original unit and omitted for the reduced units.

3 Simulating the Bulk

3.1 Periodic Boundary Conditions

The inter-atomic or -molecular interactions within any bulk material are a functionally near-infinite system. If translational symmetry is assumed, the system can be simulated by way of a repeating finite sample cell. There are N particles in this cell, but the cell is surrounded by infinite periodic images of itself. Based on the assumption of symmetry, the program enforces periodic boundary conditions (PBCs), i.e. a particle at $\underline{r}$ has a mirror image at $\underline{r} + \sum_i n_i \underline{a}_i$ for any n_i.

This eliminates the concept of a surface and means that every particle is part of the bulk.
The PBC method constructs a simulation cube with lattice vectors $\underline{a}_i$, takes the x, y, z coordinates of each particle as an array, and returns the image of that point inside the simulation cube.

Minimum Image Convention: Long-range LJ interactions are negligible, so the program applies a cutoff radius $r_c = 3\sigma$ beyond which $\underline{F}_i = 0$. For each particle, only interactions with the closest replica of another are considered.

4 Particle3D Class

The Particle3D (P3D) class generates particles in 3D space using user-defined particle names, masses, positions, and velocities.

4.1 Parameters

The following parameters are supplied to the Particle3D class by the user.
`label`: name of the particle (string)
`mass`: mass of the particle (float)
`pos`: position of the particle (float array)
`vel`: velocity of the particle (float array)

4.2 Existing Instance Methods

`__init__(self, label, mass, pos, vel)`:
Initialises a particle in 3D space.

`__str__(self)`:
Returns an XYZ-compliant string.

`kinetic_e(self)`:
Returns the kinetic energy of a P3D instance.

`momentum(self)`:
Returns the linear momentum of a P3D instance.

`update_pos_1st(self, dt)`:
Does a first order position update.

`update_pos_2nd(self, dt, f)`:
Does a second order position update.

`update_vel(self, dt, f)`:
Updates the particle's velocity.

4.3 Existing Static Methods

`new_particle(input_file)`:
Initialises a P3D instance given an input file handle.

`sys_kinetic(p3d_list)`:
Returns the kinetic energy of the whole system.
$T_{sys} = \sum_i \frac{m_i \underline{v}_i^2}{2}$

`com_velocity(p3d_list)`:
Computes the total mass and centre of mass velocity of a list of P3D instances.
$\underline{v}_{com} = \sum_i \frac{\underline{p}_{tot}}{m_{tot}}$

4.4 New Static Methods

`update_pos_2nd_list(p3d_list, box_l, dt, f)`:
Takes a P3D list and box length and does a second order position update using the Velocity Verlet algorithm. Positions follow PBCs.

`update_vel_list(p3d_list, dt, f)`:
Updates the particles' velocity in a list using the Velocity Verlet algorithm.

`msd_calc(p3d_list, p3d_list_0)`:
Calculates the MSD given the initial position P3D list and the current P3D list.

`rdf_calc(sep_list)`:
Takes the pair separations, binning them into a histogram at regular intervals.

`histogram_norm(rdf_calc)`:
Takes the histogram data from `rdf_calc` and uses `numpy` to normalise it. Returns the normalised data.

`sep_calc(p3d_list)`:
Calculates the separation between each particle in the input list and another particle using the MIC. Returns an $[N, N, 3]$ array with the pair separations and $[N, N]$ moduli.

5 `mdutilities` module

Initial conditions for particle positions and velocities are set using `mdutilities.py`. This file contains two functions which create initial conditions as follows:

Function	Input	Output
`set_initial_positions()`	`rho` - reduced particle density, `particles` - list of P3D objects	generates cubic fcc cell with N particles + returns simulation box as `numpy` array
`set_initial_velocities()`	`temp` - reduced temperature, `particles`	randomly assigns particle velocities $\underline{v}_i$ such that COM is fixed

6 Lennard-Jones module

The Lennard-Jones module uses the Velocity Verlet algorithm and static methods for time integration and computation of the total, potential, and kinetic energies of the system, inter-particle forces, MSD, and RDF.

6.1 User Input

`dt`: timestep
`numstep`: number of steps
`rho`: the reduced particle density
`temp`: the reduced temperature
N: the number of particles in the simulation box

6.2 Methods

`lj_force(`*`sep_list`*`)`:
Takes the pair separations and returns the force between particles.

`lj_pot(`*`sep_list`*`)`:
Takes the pair separations and returns the potential energy of particles interacting via the Lennard-Jones potential.

`main()`:
The main method calls on `Particle3D` to create N P3D instances. It then passes a list of these instances to `mdutilities`. It uses a time integration loop to update the forces, positions, velocities, and energies of the particles. It appends each of these quantities to separate data lists. There will be an initial position list and an updated position list, named `p3d_list_0` and `p3d_list`, respectively. The former is necessary to calculate the MSD. There will also be a list of pair separations, `sep_list`. Finally, the main method writes an XYZ-compatible trajectory file that is visualised using VMD.