Python Final Project - Lennard Jones Molecular Dynamics Simulations

In Chem 280 (the prerequisite for this course) we wrote a Monte Carlo simulation code for a Lennard Jones fluid. In this class, we have used this MC code to study a system under varying conditions (lab 1), performed molecular dynamics simulations using research softare (lab 2), and wrote our own MD integrators for a diatomic system (problem set 2).

This repository contains an implementation of a molecular dynamics simulation for a Lennard Jones fluid. The provided simulation runs at constant number of particles (N), constance volume (V), and constant energy (E). Your task for this project will be to assess the code that is been provided, and improve the its structure and performance. You should improve the code performance by examining the implementation, looking for unneccessary calculations, and through profiling the code and making appropriate changes (using `snakeviz`, for example). This simple MD implementation only uses the Python Standard Library (except for plotting with matplotlib). You may used pandas or NumPy in your improved version.

You can run the simple molecular dynamics simulation by using the following command in your terminal

```
python simple_md.py
```

The MD simulation will output a graph showing total energy vs time and an `xyz` file which contains the simulation trajectory (you can use a program like VMD to visualize this, or you might be able to use NGLView to view it in a Jupyter notebook.)

Molecular Dynamics - The Procedure

The procedure for performing a molecular dynamics simulation is outlined in this section.

The MD code you have been provided with is based on the book "Understanding Molecular Simulation" Chapter 4 (available through UC Berkeley Libraries). The procedure for a molecular dynamics simulation is

1. Read in the parameters that specify the conditions of the run (initial temperature, number of particles, density, time step)
2. Initialize system using parameters - set initial positions and velocities of particles.
3. Compute the forces on all particles.
4. Integrate Newton's equations of motion. This will use an integrator like the Velocity Verlet integrator. This step and the previous step make up the core of the simulation, and will be repeated until the simulation is finished.
5. Analyze the simulation - compute average of measured quantities or analyze the generated trajectory.

Pseudo Code for MD Loop

```
initialize()

for i in range(num_steps):
    calculate_force()
    integrate()
    t = t + timestep

analyze()
```

Building from the Monte Carlo Simulation

Molecular dynamics requires many of the same ingredients we used in our Monte Carlo simulation. In the Monte Carlo simulation, we measured the distance betwen particles and calculated the potential energy. For our molecular dynamics simulation, we will also need to measure the particle-particle distance to calculate the force. A notable difference between MD and MC is that for MD, we will need to move the particles in the X, Y, and Z direction according to forces in those directions. This means we will need the distance **vector** (rather than the magnitude) and the force **vector**.

Recall from our lab on MD and problem set 2 that force is the negative gradient of the potential energy:

$$\vec{F} = -\nabla U$$

For this simple simulation, we will use particles that interact with the Lennard Jones potential:

$$U^*\left(r_{ij}\right) = 4\left[\left(\frac{1}{r_{ij}^*}\right)^{12} - \left(\frac{1}{r_{ij}^*}\right)^{6}\right]$$

For the molecular dynamics simulation in three dimensions, we will need a force component in each direction, x, y, and z, meaning that we will also need the distance vector.

The force component in each direction can be obtained by finding the magnitude of the force (derivative of the LJ potential), then multiplying it by the unit vector of the distance. The force component in the x direction is equal to

$$f_x(r) = \frac{48x}{r^2}\left(\frac{1}{r^{12}} - \frac{1}{2r^6}\right)$$

This is performed for X, Y, and Z.

Simulation Initialization

To start the simulation, we will need to assign initial positions and velocities to all particles in the system.

For our molecular dynamics simulations, we will use `generate_cubic_lattice` to generate the initial coordinates. Generating random coordinates is not appropriate for molecular dynamics simulations as the simulations are very sensitive to starting conditions, an overlapping particles will cause simulation instability.

To set the initial velocities of the system, we consider the specified initial temperature. Velocity and temperature are connected through kinetic energy. For the average kinetic energy per degree of freedom, temperature and kinetic energy are related

$$\langle \frac{1}{2} m v^2 \rangle = \frac{1}{2} k_B T$$

The temperature at a particular time-step during the simulation can be calculated from the velocity of the particles.

$$T(t) = \sum_{i=1}^{N} \frac{m_i v_i^2(t)}{k_B N_f}$$

where is the number of degrees of freedom for a system of particles, usually approximated to $3N$ in three dimensions.

In practice the system is set to the target temperature initially by assigning a random velocity on the range of `-0.5` to `0.5` to each particle, then scaling the calculated velocities using a calculated scaling factor in order to set the simulation temperature to the desired value.

$$sf = \sqrt{\frac{3T}{\sum\limits_{i=1}^{N} v_i^2}}$$

This is implemented in the `set_initial_velocities` function. Note that the temperature will change during this simulation, because we are running an NVE (constant number of particles (N), volume (V), and energy (E)) simulation.

Integration

This MD simulation uses the Velocity Verlet algorithm, which was covered in Problem Set 2.

Improving the Simulation

Your task for this project is to improve the molecular dynamics simulation script which has been provided in this repository. You will need to do the following:

1. Evaluate and improve code performance. You should consider the necessity of each calculation and how you can leverage third party libraries, such as NumPy or Pandas in your software.
2. Improve the project structure - You should examine the code structure and consider ways to improve its structure. You might choose to change function defintions or to implement classes. This should include adding appropriate error checks.

3. Improve the project documentation - make sure the project is adequately documented including in-code (comments and docstrings) and out of code (README) documentation.
4. Running the project - Your simulation should have a command line interface. You should be able to run from the command line by inputting a simulation temperature, number of atoms, target density, output file, and options for creating graphs of the total energy or temperature vs simulation time step. Your command line interface should also have a `--help` message. Your command line interface should be documented in your `README`.
5. You should use at least one example of inheritance, at least one example of a decorator, and at least one example of a context manager in your code.

Turning in Your Assignment

Consider the project you turn in to be a software product. Include files such as a `.gitignore` and be careful with the files you commit. Your repository should have a `README` describing the project and its installation. You should also write in detail about any optimization or design decisions you made for improving the program. Please be very descriptive and verbose when explaining the decisions you made.