Simulating Particle Motion in a Lennard-Jones Potential Using a Monte Carlo Simulation With the Metropolis Algorithm

Brenda Angulo, Ryan Chin, Anika Galvan, Chandler Goodwin, Zander Rothering

Department of Chemistry, College of Chemistry, University of California at Berkeley

---

Our group simulated the motion of a particle using a Metropolis Monte Carlo model with a Lennard-Jones-style pair potential in both a Python class package and C++ for performance comparison. The simulation uses the Metropolis algorithm, where trial moves are proposed for individual particles and accepted or rejected based on the change in potential energy and the Boltzmann acceptance criterion.

The Lennard-Jones potential describes how two particles interact depending on their separation r. In the simulation this gives the energy cost/benefit of placing particles at certain distances from each other. This is found by calculating the change in potential energy (ΔE) caused by the proposed move.

The Boltzmann factor sets the acceptance criteria of a proposed particle move in the Metropolis Monte Carlo algorithm. If the energy goes down, accept the move; if it goes up, the move is accepted with a probability that depends on temperature.

This generates a sequence of moves distributed according to the Boltzmann distribution for the Lennard-Jones system, and ensures the simulation is thermodynamically consistent.

The python code allowed for quick testing and optimization of different parameters. C++ code improves speed. Both have similar results.

Contents

---

1 Introduction

1.1 Purpose and History of the Lennard-Jones Potential

With the goal of simplifying the need for complicated integrals in characterizing pair potentials (U) in mind, Lennard-Jones proposed a polynomial which included an attractive and repulsive term:

$$U(r) = \frac{a}{r^{12}} - \frac{b}{r^{6}}$$

Where r is the distance separating the particles, center to center. Constants a and b are defined respectively:

$$a = 4\epsilon\sigma^{12}$$

$$b = 4\epsilon\sigma^{6}$$

Where ϵ describes the attractive force and σ defines the distance at which the potential is equal to 0. The Lennard-Jones (12, 6) has become the most commonly used LJ potential since its adoption in 1931.[1,2]

1.2 The Monte Carlo

As with the Lennard-Jones potential, the Monte Carlo method worked to pivot away from complex mathematical problems in order to determine probabilities in uncertain processes.[3] With computational simulation in mind, the Monte Carlo model is straightforward and in turn faster to compute when compared to molecular dynamics.[4] In a Lennard-Jones potential which uses the Monte Carlo model, simulated atoms move randomly, with each move accepted or rejected based on the change in temperature and energy. This acceptance/rejection criteria describes a specific type of Monte Carlo model- the Metropolis algorithm.

1.3 Metropolis Algorithm

Shortly after the publication of the Monte Carlo method, Arianna Rosenbluth and Nicholas Metropolis formulated the Metropolis algorithm, enabling the ability to create a sample from complex probability distributions.[5] The probability distribution being modeled is used to determine whether each move is accepted or rejected. While exploring the probability distribution in a stepwise manner, also known as a random walk, can require many iterations, it is particularly useful in distributions of higher dimensions.

To computationally simulate the motion of many particles in a controlled environment, a Monte Carlo method with the Metropolis algorithm will be used to observe the Lennard-Jones potential of each particle and its subsequent movement behavior. The particle movement will be influenced by brownian motion as well as the Boltzmann factor.

2 Methods

2.1 Equations and Algorithms

I. Lennard-Jones Potential

$$U(r) = \frac{a}{r^{12}} - \frac{b}{r^{6}}$$

In this simulation, the Lennard-Jones Potential of each particle is determined at each iteration, accounting for the interactions with all of the surrounding particles in this many particle system.

II. Monte Carlo for Random Sampling

In order to determine where each particle may move, the simulation follows the Monte Carlo method for randomized decision making. For this case, the Monte Carlo method will simulate particle movement as well as the probability of Brownian Motion.

III. Boltzmann Constant

While the Lennard-Jones Potential is used to determine the potential energy of each particle, another condition for the Metropolis algorithm will be based on the Boltzmann probability.

$$p_i \propto \exp\left(-\frac{\varepsilon_i}{kT}\right)$$

Where k is the Boltzmann constant:

$$k_b = 1.380649 \times 10^{-23} J/K$$

The Boltzmann Distribution demonstrates the probability (pi) that a particle will exist in a given state with energy (Ei) given environmental temperature (T) and Boltzmann constant. With this probability, we will determine the probability of a particle moving even if the potential energy is increasing.

IV. Metropolis Algorithm

While the Monte Carlo method is used for determining possible positions for each particle, the Metropolis algorithm will work to accept/reject each possible move after evaluating the energies against certain criteria.

3 Implementation

3.1 Python Code

The simulation script defines 5 functions: Plot_Particles, LJ_Potential, _Ei, CoordstoPotential, Utot_Move_Particle. In order to run the simulation, the class ParticleMotion is called, where a method Run_Particle_Movement is called to run through N iterations. The functions are described as follows:

a. Plot_Particles

In order to plot the current particle positions at Nth iteration, Plot_Particles is called, taking the x and y coordinates of each particle. The randomly generated mass for each particle is represented by the marker size.

b. LJ_Potential

The Lennard-Jones Potential of a particle is calculated when called.

c. _Ei

With the Lennard-Jones Potential, the total potential energy a particle has, based on all of the surrounding particles, is summed up using vector math. Numpy method np.inf is used to prevent dividing by zero (the particle's distance from itself)

d. CoordstoPotential

The goal of CoordstoPotential is to compute the total potential energy of all of the particles using broadcasting. This function also avoids self interaction with the Numpy method np.fill_diagonal.

By generating a distance matrix, r^2, the Lennard-Jones Potential can be calculated and summed for the entire system.

e. Utot_Move_Particle

Using the functions above, Utot_Move_Particle uses the Monte Carlo method to generate random particle movements with np.random.choice. After determining the new positions of each particle, the new Lennard_Jones Potentials are calculated. The Metropolis algorithm is demonstrated by determining if the particle gets to move to its new position under specific conditions. If the potential energy decreases, the particle is allowed to move. If the potential energy increases, the particle may move depending on the Boltzmann probability, based on Brownian motion and the temperature of the system.

3.3 C++ Code

While functionally similar, our C++ script cannot rely on the numpy library as we do in Python. Therefore we must utilize a function for determining the potential energy of the particles in our system. When running our file, the main function will run, which then calls the CoordsToPotential function twice at each time step - once to determine the energy of the system with all proposed moves, and then to determine it once the moves are cherry picked.

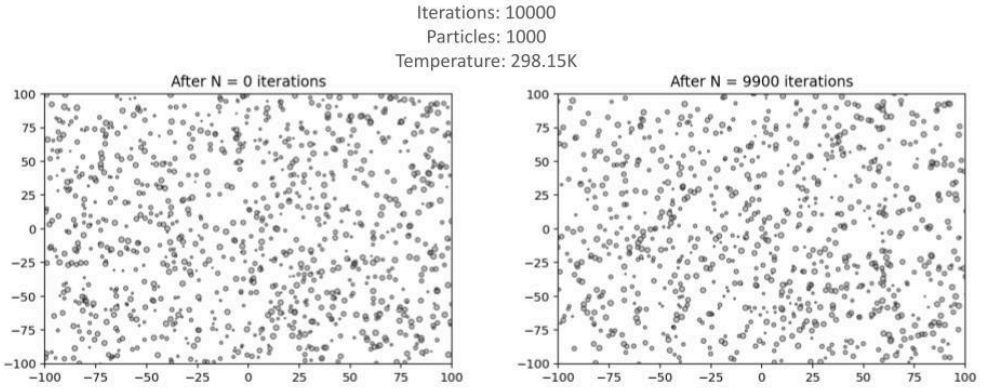
a. CoordsToPotential
   Similarly to our Python code, this function serves to generate a vector containing the potential of each of our particles at any given position. Once we have this vector, we can use it to compare against potentials at the position of the previous step. This function takes arguments in the form of vectors for x position and y position, doubles for constants a and b, and an integer for the number of particles in our potential vector.

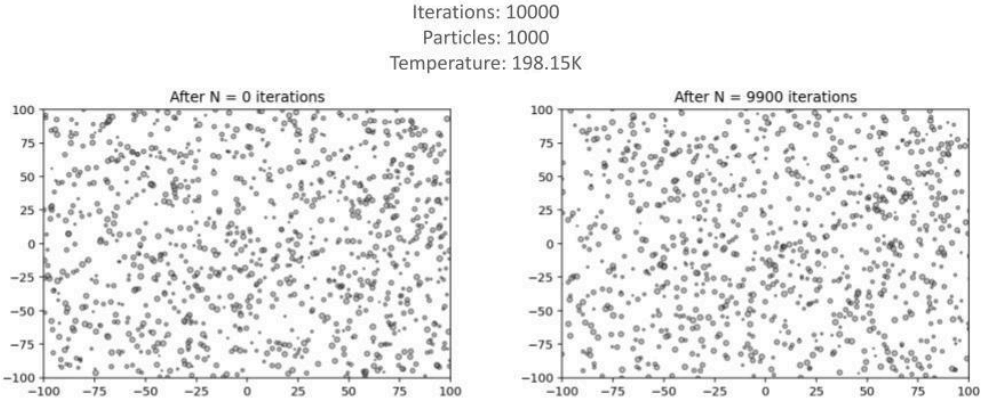
b. main
   Our main function serves to run the simulation after defining the necessary functions for computation of Lennard-Jones potential. The function defines the variables, vectors, and random distributions required to calculate the Lennard-Jones potential before calling the function CoordsToPotential twice to calculate the potential before and after moving each particle. The move is then accepted based on if the potential decreased. For moves that are not accepted, particles are given an opportunity to undergo Brownian motion, determined by comparing the particle's Boltzmann factor with a randomly generated probability.

4 Evaluation

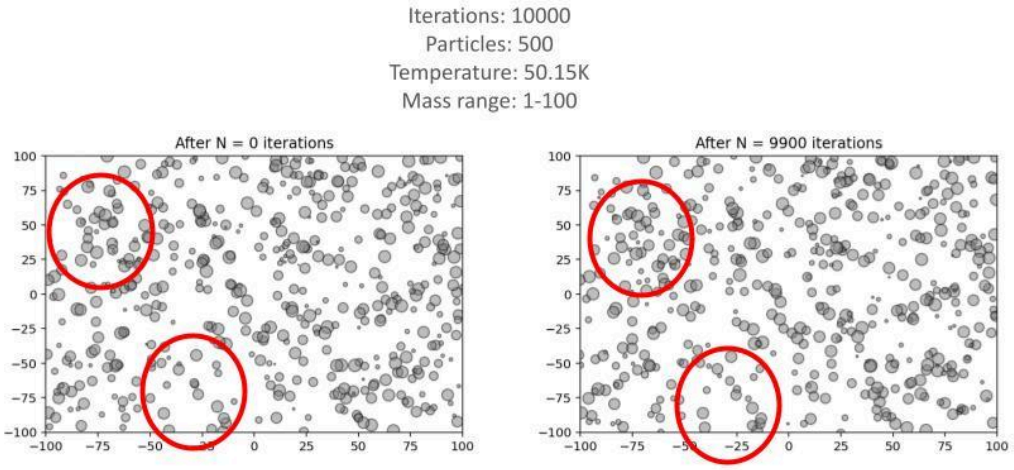
4.1 Clustering vs Temperature and Mass

While observing the evolution of the plots, the most noticeable change is the clustering that occurs. This clustering is representative of the particles settling into pockets of low energy.

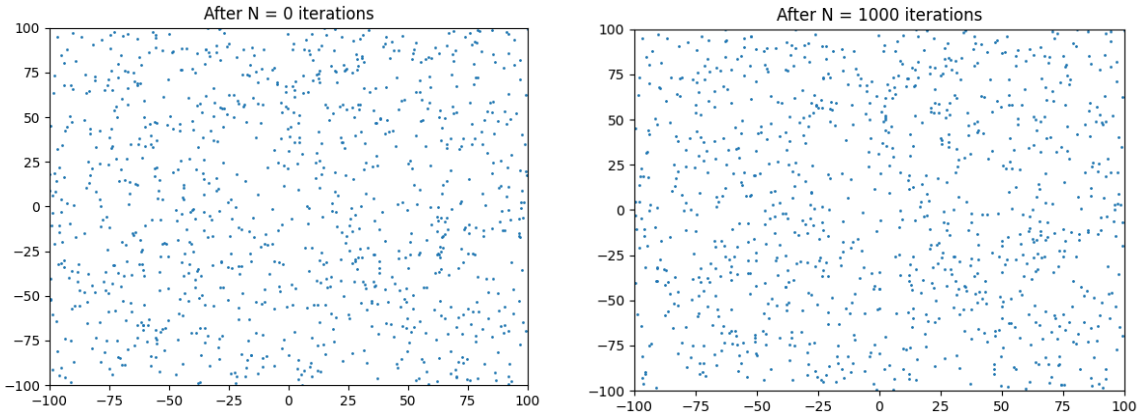




At lower temperatures, it is theoretically expected that larger clusters would form. We also experimented with a larger range of mass to observe any differences in clustering behavior. In the case where the particle mass was larger, clustering became clearer and more noticeable in the final iteration.



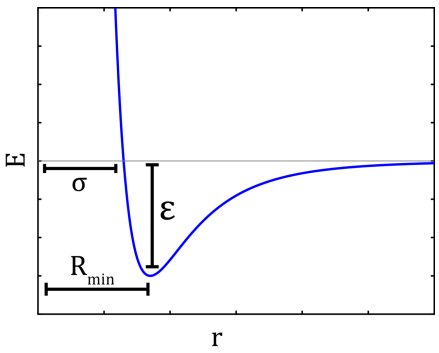
When running the C++ script, we are able to see similar behavior between particles. Given more time, we would have liked to fine tune our plots in C++ to show the mass of each particle as well, as it becomes much easier to see where the clustering is happening. However, given that our greater proficiency is with Python, we have chosen to showcase the majority of our findings in the plots above.



5 Discussion

5.1 Interpretation of Results

Shown here is a plot of the expected potential energy (E) on the y-axis vs the distance (r) on the x-axis, where sigma is the distance between the two particles where E(r)=0. We expect particles to naturally gravitate toward surrounding particles and generally maintain a distance of Rmin, as this is where our potential is minimized.[6]



In our plots, we see this manifest as the clustering of particles around each other at distances we expect to be approximately Rmin. Notably, the Rmin for each particle in relation to its neighbors is not directly affected by the mass of our particles. However mass does affect the particle's random movement as particles with higher masses are subject to proportionally smaller Brownian movements.

5.2 Limitations of the Model

The primary limitations of the present model are its simplified representation of physical reality and its limited scalability due to computational complexity. The simulation is restricted to a two-dimensional space in which particles are permitted to spatially overlap, rather than employing a three-dimensional hard-sphere that enforces excluded volume constraints. From a computational standpoint, the current implementation evaluates interactions between every particle pair at each time step, resulting in computation cost. Consequently, as the number of particles and simulation steps increases, the computational demands grow, imposing practical limits on the achievable size of the system.

5.3 Proposed Improvements

Future work could extend the simulation framework to incorporate multiple particle species with distinct physical properties. Presently, particles' differentiation is limited to the differences in mass; however, additional characteristics such as electric charge could be implemented, enabling the modeling of electrostatic interactions, such as Coulombic attraction and repulsion. Another enhancement would be the inclusion of external fields, such as gravitational or electromagnetic fields, that influence particle trajectories. This would require the implementation of spatially varying potential gradients, with particles accelerating along or against the gradient depending on interaction with the field. Incorporating full Newtonian dynamics would further improve physical realism by allowing particles to retain velocity between time steps, thereby conserving momentum, in contrast to the current random step-based motion. Additionally, the model presently represents particle movement in two spatial dimensions plus a single dimension of time. Extending the spatial dimensions to three would allow for a more physically accurate treatment of particle trajectories and the enforcement of particle volume excluding particles from overlapping in 3D space. Collectively, these enhancements would yield a simulation that more closely reflects the underlying physics of particle motion.

6 References

1. Moscato, P., & Haque, M. N. (2024). New alternatives to the Lennard-Jones potential. *Scientific Reports*, *14*(1). https://doi.org/10.1038/s41598-024-60835-8
2. Lenhard, J., Stephan, S., & Hasse, H. (2024). On the history of the lennard‐jones potential. *Annalen Der Physik*, *536*(6). https://doi.org/10.1002/andp.202400115
3. McDonald, I. R., & Singer, K. (1969). Examination of the adequacy of the 12–6 potential for liquid argon by means of Monte Carlo calculations. *The Journal of Chemical Physics*, *50*(6), 2308–2315. https://doi.org/10.1063/1.1671381
4. *Monte Carlo Lennard-Jones*. NetLogo Models Library: Monte Carlo Lennard-Jones. (n.d.). https://ccl.northwestern.edu/netlogo/models/MonteCarloLennard-Jones
5. Betancourt, M. (2018). The convergence of Markov chain Monte Carlo methods: From the metropolis method to Hamiltonian Monte Carlo. *Annalen Der Physik*, *531*(3). https://doi.org/10.1002/andp.201700214
6. *File:12-6-lennard-jones-potential.SVG*. Wikimedia Commons. (n.d.). https://commons.wikimedia.org/wiki/File:12-6-Lennard-Jones-Potential.svg