The Lennard-Jones Potential and the N Body Problem

Archie Duffy
School of Physics, University of Bristol.

(Dated: April 24, 2024)

This paper outlines a basic n-body code relying on the Lennard-Jones potential developed in Python to facilitate an intuition of the n-body problem and the Lennard-Jones potential. The code is tested to show: particles oscillating, bound by potential and elastic interactions with non-periodic boundary conditions. Then a series of tests are carried out with the code to examine the dependence of the system's pressure on the number of particles and temperature.

INTRODUCTION

The history of n-body simulation is marked by a progression from early conceptualisation to modern computational sophistication. Initially rooted in Isaac Newton's laws of motion[1], analytical solutions to N-body problems emerged, albeit with limitations due to computational constraints. The advent of computers in the mid-20th century ushered in a new era, enabling researchers to explore numerical methods for simulating the dynamics of multiple interacting bodies.

In 1924, Lennard-Jones (at the time known as Jones) released a series of articles titled "On the determination of molecular fields"[2], laying the groundwork for molecular interaction studies. Schrödinger's 1926 quantum mechanics formulation[3] and London's 1930 work[4] built upon this foundation to derive an approximate solution for dispersion forces. In 1931, Lennard-Jones proposed the widely used effective pair potential[5], known as the Lennard-Jones (LJ) potential, which describes molecular interactions based on parameters like distance, well depth, and zero-crossing distance. This LJ potential has since been instrumental in understanding molecular-level phenomena.

The introduction of the Verlet integration method by Loup Verlet in 1967 [6] revolutionised computational simulations, particularly in molecular dynamics and N-body simulations. This method provided a simple yet powerful numerical technique for solving Newton's equations of motion with remarkable accuracy and computational efficiency. Its straightforward approach updates particle positions and velocities based on previous positions, current accelerations, and a time step, reducing computational overhead and enhancing numerical stability. Alongside, tree-based methods like the Barnes-Hut algorithm [7] further improved the scalability of n-body simulations, enabling efficient calculations of gravitational forces between distant groups of particles. As a result, N-body simulations became invaluable tools across various fields, including astrophysics, cosmology, molecular dynamics, and condensed matter physics, facilitating the study of complex physical phenomena where analytical solutions are challenging to achieve.

With the rise of high-performance computing (HPC) systems, researchers can perform exceptionally large-scale n-body simulations with millions or even billions of particles. However, challenges persist, including computational efficiency, numerical accuracy, and the need for realistic physical models. Despite these challenges, ongoing research continues to push the boundaries of N-body simulation capabilities, driving further innovation and insight into the dynamics of complex systems.

The n-body simulation that this paper analyses is aimed at undergraduates as a tool for understanding the theory behind the n-body problem and facilitating an intuition of the evolution of n-body systems. It is not the most sophisticated model, as it uses Python to emphasise readability and make the simulation accessible to a wider audience. Python is one of the most popular languages, particularly amongst undergraduates.

1 - BASIC N-BODY CODE

In the N-body simulation, trajectories were established for a fixed number of particles N. Each particle, identified as i, had an associated position vector $\mathbf{r}_i$ and velocity vector $\mathbf{v}_i$ in 3D space. Interactions between particles were mediated through the forces exerted between them as given by the Lennard-Jones force, derived by taking the negative gradient of the Lennard-Jones potential:

$$V(r) = 4\epsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right), \tag{1}$$

where $r = |\mathbf{r}_j - \mathbf{r}_p|$ is the separation between two atoms located at $\mathbf{r}_j$ and $\mathbf{r}_p$, and ϵ and σ are parameters that depend on the atoms. For this particular simulation an ϵ value of 125.7 $\times k_B$ and a σ value of 0.3345 $\times 10^{-9}$ m corresponding to the values of an Argon atom. The Lennard-Jones force is then:

$$F_{ij} = -\nabla V_{\text{LJ}}(r) = \frac{24\epsilon}{\sigma}\left(-2\left(\frac{\sigma}{r}\right)^{13} + \left(\frac{\sigma}{r}\right)^{7}\right)\hat{r}_{ji}, \tag{2}$$

where

$$\hat{\mathbf{r}}_{ji} = \frac{\mathbf{r}_j - \mathbf{r}_i}{|\mathbf{r}_j - \mathbf{r}_i|}, \tag{3}$$

is a unit vector pointing from atom i to atom j.

The net force acting on particle i, denoted as $\mathbf{F}_{\text{net}}$, was computed by summing the forces from all other particles (excluding i), resulting in:

$$\mathbf{F}_{\text{net}} = \sum_{j\neq i}\mathbf{F}_{ij}. \tag{4}$$

The acceleration of particle i,$\mathbf{a}_i$, was then determined using Newton's second law as:

$$\mathbf{a}_i = \frac{1}{m_i}\sum_{j\neq i}\mathbf{F}_{ij}, \tag{5}$$

where m_i denoted the mass of particle i. These accelerations updated velocities and positions over a specified time step δt.

Then, the Verlet 'leap-frog' integration scheme was implemented. For each particle i, velocities were updated halfway through the time step using the formula:

$$v_i = v_i + \frac{\delta t}{2}a_i. \tag{6}$$

Then, these updated velocities were used to adjust the positions over the entire time interval with:

$$r_i = r_i + \delta t v_i. \tag{7}$$

Finally, the new positions were used to recalculate the accelerations $\mathbf{a}_i$ with equation 5, followed by the calculation of new velocities after the remaining half of the time step using equation 6 again.

To make the code as efficient as possible within Python's limitations, array operations were used in place of for loops where possible. This speeds up the code, as each element can be operated on simultaneously instead of one at a time.

To further simplify the code, a conversion into dimensionless units was completed. This was conducted with the following equations:

$$r = \frac{\text{length}}{\sigma}, \quad m = \frac{\text{mass}}{m_a}, \quad t = \frac{\text{time}}{\tau}, \tag{8}$$

which uses characteristic length, σ, and time, τ, of interactions for argon atoms. The characteristic time is given by relating other characteristic variables in the equation:

$$\tau = \sqrt{\frac{m_a\sigma^2}{\epsilon}}. \tag{9}$$

From this, calculated energy values will be measured relative to ϵ; accelerations are measured relative to $\frac{\sigma}{\tau^2} = \frac{\epsilon}{m_a\sigma}$ and speeds are measured relative to $\frac{\sigma}{\tau} = \frac{\epsilon}{m_a}$.

After implementing all the aforementioned equations and unit conversions, a simple test was conducted to confirm if the code performed as expected. This involved placing two particles one σ apart and observing their interaction. The equations predict that the two particles should oscillate. This can be seen in the two terms of equation 2. At small r values, the repulsive first term dominates. At large r values, the attractive second term dominates, resulting in bound oscillating particles.

Figure 1 shows two plots, energy and kinematic variables, for the particle that started at the origin in the simulation. These were constructed by calculating the kinetic energy using:

$$E_k = \frac{1}{2}\sum_{i=1}^{N} m_i v_i^2, \tag{10}$$

where m_i and v_i are the mass and velocity of the ith particles and the potential energy with equation 1 at each time step for each particle. These values and the kinematic variable values, which were calculated in the Verlet integration, were added to an array at each time step and then plotted against time. Additionally, a live plotting feature was added, which allowed the trace of the particle and the recorded values to be plotted while the code was being executed as a diagnostic tool.

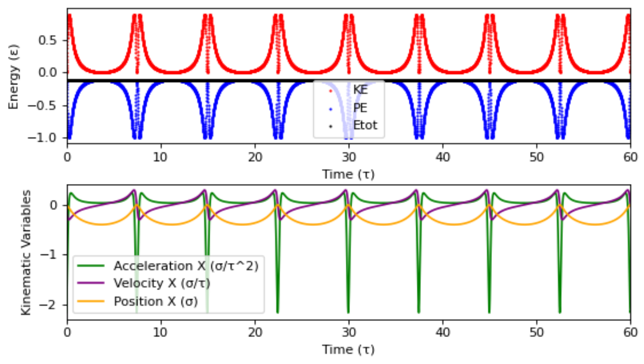


FIG. 1: Energies; kinematic variables and history of a bound particle due to the Lennard-Jones potential. Corresponding to figure 2

From figure 1 it can be seen that the particle is bound and oscillating between zero and minus one as predicted in the theory. Figure 2 shows the particle in dark blue and the entire history in light blue. The trace shows the same oscillation of the left particle between zero and minus one in a characteristic straight line.

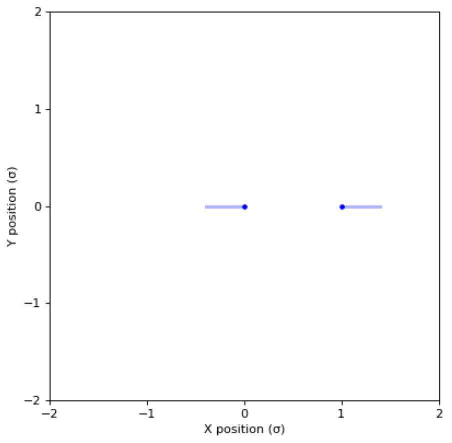


FIG. 2: Trace of two bound particles starting at rest one σ apart. Corresponding to data seen in figure 1.

From figures 1 and 2 it can be concluded that the basic n-body code functions as expected and further features could be added to the code knowing that there was a solid foundation to work from.

2 - BOUNDARY CONDITIONS

In coding simulations, boundary conditions dictate how particles behave when interacting with the simulation domain's edges. There are two main types of boundary conditions: periodic and non-periodic.

Periodic boundary conditions involve particles re-entering the simulation domain from the opposite side when they exit, creating a seamless transition as if the domain were wrapped around like a torus. From a coding perspective, this means adjusting particle positions so that they reappear on the opposite side when they move beyond the boundary. This ensures continuous interactions among particles across the boundaries as if the simulation domain were infinitely repeating.

On the other hand, non-periodic boundary conditions impose constraints on particle movement at the boundaries, such as reflecting or absorbing particles upon contact with the boundary. For instance, particles might bounce off the boundary with an opposite velocity vector, simulating a physical wall. In coding, this requires explicit checks to detect when particles cross the boundary, followed by appropriate actions to handle the interaction, like reversing velocities or removing particles from the simulation.

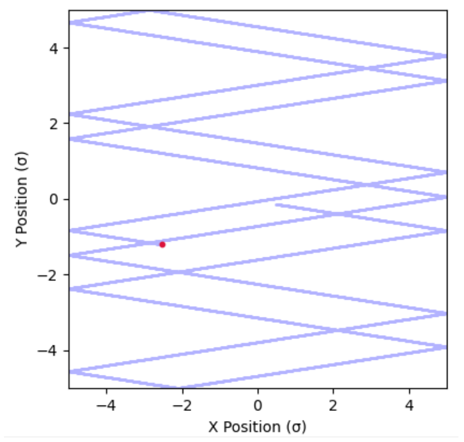


FIG. 3: Particle bouncing with non-periodic boundary conditions. The particle is in red, and the trace is in light blue. Corresponding to figure 4.

For this n-body code, non-periodic boundary conditions were utilised as interactions between the walls and the particles are to be studied, simulating a system like gas in a bottle. Periodic boundary conditions can be useful in situations where the interactions should be uninterrupted across the boundary, for example, simulating an interesting piece of a larger system such as fluid dynamics.

Boundary conditions were implemented into this code with four steps. During the Verlet integration, after the first position update, a copy was made of each particle position. Each boundary was then checked to see if the particle would leave the boundary after the next time step of velocity. If any particle exceeded the boundary, its velocity would be reversed by multiplying it by minus one. This was carried out component-wise. For example, if the particle exceeded the box size in the y direction, only the y component of its velocity would be reversed. The particle would then be set to its previous position if it did exceeded the boundary. The result is that the particle stays within the boundary and bounces elastically.

A test case with the box was executed to verify that the code worked as expected. This was conducted using a single particle with random initial position and velocity. The trace from this simulation can be seen in figure 3 where the particle bounces off the walls as expected.

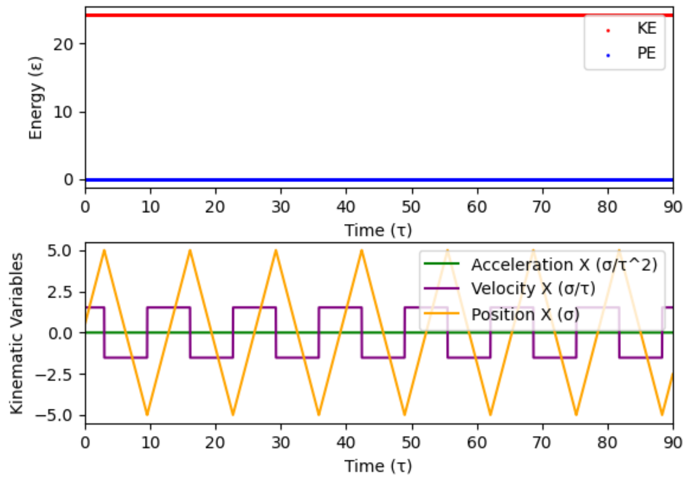


FIG. 4: Variables from a ball bouncing in a box with kinematic variables. Corresponding to figure 3.

Further evidence for a successful implementation of the boundary conditions is shown in figure 4 where energy and kinematic variable traces are shown. The first plot shows constant kinetic and potential energy values. This shows that the magnitude of the velocity (used in calculating the kinetic energy) is constant as would be expected for the correct implementation of boundary conditions as only the direction changes on collision with the boundary.

Additionally, the second graph in figure 4 shows zero acceleration, a square wave for the velocity in the x direction, and a triangle wave for the position in the x direction. This shows that the velocity remains unchanged except at the boundary, where it changes instantly to the same value multiplied by minus one as coded. This occurs at the boundary as can be seen by the position line reaching the box boundary of five as the velocity changes direction.

3 - TEMPERATURE, PRESSURE AND EXPERIMENTATION

At this point, there was enough confidence in the basic code to progress onto testing the effect of changing parameters and observing the results. The recorded data included the temperature and pressure values calculated from the particles. The temperature was calculated from the total kinetic energy of the system using the equation:

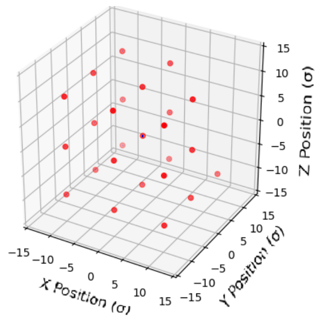
$$E_k = \frac{1}{2}\sum_{i=1}^{N} m_i v_i^2 = \frac{3}{2} N k_B T, \tag{11}$$

where E_k, M_i and v_i are as outlined in equation 10, N is the number of particles k_B is the Boltzmann constant and T is the temperature. For the sake of simplicity, the Boltzmann constant was set to unity, therefore defining a temperature of $1k = 1.380649 \times 10^{-23} J$ as it is directly proportional to the energy.

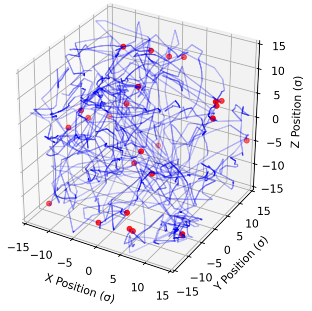
The pressure calculation was more complicated as it involved imagining an area A in the $y-z$ plane. For any particle that crosses this plane in a time Δt the x-component of the momentum is taken, $p_x = mv_x$, where v_x is the speed of the particle in the x-direction. The pressure in the x-direction is then:

$$P_x = \frac{1}{A\Delta t} \sum_{i \text{ crossing}} m_i v_{xi}, \tag{12}$$

where the sum is performed only over the particles i crossing the plane in the time interval Δt. Assuming isotropic pressure, $P_x = P_y = P_z = \frac{1}{3}P$, where P is the total pressure.



(a)$t = 0\tau$



(b)$t = 30\tau$.

FIG. 5: 3D plot of 25 particles.

The number of particles was increased for further testing to try and reduce the stochastic nature of systems with few particles. To facilitate this, a grid initialisation was implemented to prevent two particles from being placed close together and causing large forces at the start of the simulation. To check the particles were initialised in a grid correctly and for further diagnostics, a 3D plotting function was added to the code. Figure 5(a) shows the plot on initialisation and figure 5(b) shows an example evolution for particles initialised on a grid.

Even with grid initialisation, there were still some cases of large forces during the initial few τ of the simulation. Figure 6 shows an example of this.

To get the pressure at equilibrium and ignore the outliers caused by the initialisation the first 10 τ of the simulation were ignored when determining pressure. This resulted in a much more stable pressure as seen in figure 7.

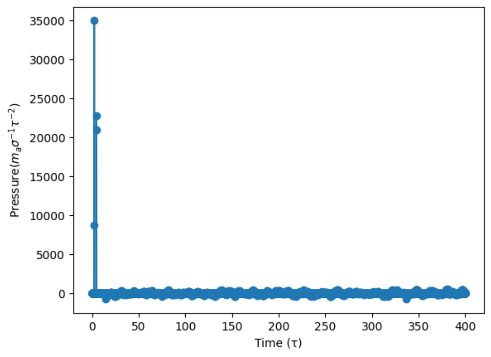


FIG. 6: Pressure readout from a simulation of 70 particles initialised on a grid with identical initial velocity.

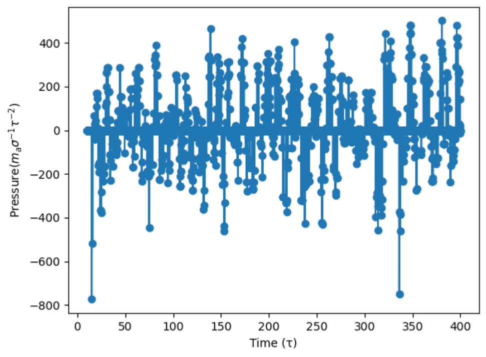
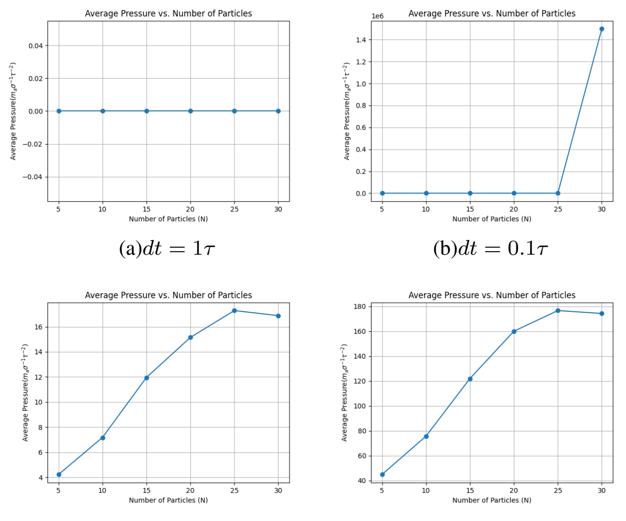


FIG. 7: Figure 6 with the first 10 τ of data omitted.

Before further testing, it was verified that an isotropic pressure assumption was valid. This was achieved using the same method of calculating pressure in the x direction in both the y and z directions. As can be deduced from figure 9 an isotropic model is a fair simplification as each pressure value is independently calculated close to one another with pressure values of $Px = 45.291 \pm 0.008, Px = 47.42 \pm 0.008, Px = 47.157 \pm 0.008$.

Additionally, the simulation parameters were picked. To do this a series of tests were conducted. To choose the time step, a simulation of 400 τ was performed using different dt. The data can be seen in 8. From this, time steps of $dt = 1\tau$ and $dt = 0.1\tau$ can be seen to be no good for a simulation. Time steps $dt = 0.01\tau$ and $dt = 0.001\tau$ are closer to what might be expected of a simulation such as this where a linear dependence is predicted with little difference between the two plots. As there was minimal difference between the two and the simulation with $dt = 0.001\tau$ took around ten times longer (20 minutes compared to 2 for the smaller time step) a dt of 0.01τ was used moving forward.

Similar tests to those carried out for the determination of dt



(a)$dt = 1\tau$ (b)$dt = 0.1\tau$

(c)$dt = 0.01\tau$ (d)$dt = 0.001\tau$

FIG. 8: How the pressure variance with N changes with the time step, dt, used in the simulation

were conducted for the number of particles; total simulation time and initial conditions. Initial velocities on the order of the length scale of the system were found to give good results and as such 2.5 $\sigma\tau^{-1}$ was used as the standard initial velocity. An N value of around 50 was found to give reasonably high-resolution results within a manageable time frame. The total time of the simulation was taken in a similar way to that of the dt value as a time when systems approach equilibrium within the limitations of the discrete nature of the simulation and without increasing computational demand to unreasonable levels.

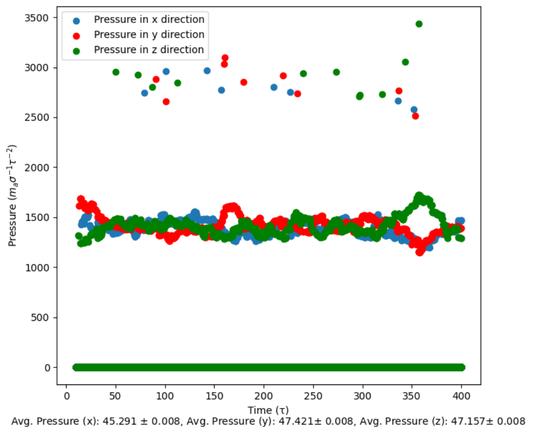


FIG. 9: Pressure comparison in x, y and z directions.

A real test of pressure dependency on the number of particles was then conducted. This used: grid initialisation; a total simulation time of 800 τ; a time step of 0.01 τ; a box size of 30 σ and a uniform initial velocity of 2.5 $\sigma\tau^{-1}$. The results, shown in figure 10, demonstrate an almost linear dependence of pressure on the number of particles. This could be predicted from 12 outlining the pressure for one particle, which is then scaled linearly over the system. Additionally, it could be considered in terms of the Van Der Waals equation relating pressure and number of particles is:

$$p = \frac{RT}{v-b} - \frac{a}{v^2}, \tag{13}$$

where p is pressure, R is the ideal gas constant, T is temperature, v is the molar volume ($v = \frac{V}{N_A/N}$) and a and b are system dependent constants. In the limit where $v \gg b$ and and when $v \gg \left(\frac{a}{p}\right)^{1/2}$ then the equation simplifies to:

$$pV = NkT. \tag{14}$$

This provides more evidence for the direct proportionality between pressure and number of particles. There are slight deviations from a perfect linear fit, which can be explained by the discrete nature of the simulation and the limitations on computing power and time available to run the simulation.

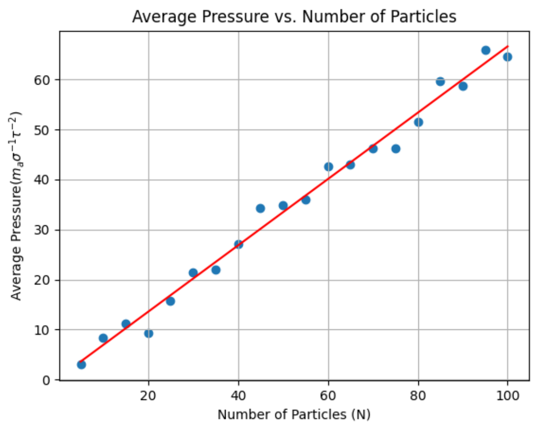


FIG. 10: The dependence of pressure on the number of particles in the system.

Next, the effect of temperature on the pressure was investigated by changing the velocity value and calculating the temperature and pressure to observe the dependence. This used a total time of 800 τ; a time step of 0.01 τ; a box size of 30 σ; 50 particles and uniform velocity which was varied between 0.1 $\sigma\tau^{-1}$ and 2.5 $\sigma\tau^{-1}$. As seen in figure 11 the pressure has an almost linear dependence on temperature as predicted by Gay-Lussac's law[9].

The final test completed with the code was the dependence of pressure on volume. This used the same initial conditions as in the particle number dependence simulation, except that the total volume was varied by changing the box size between 10 and 100 σ and used a constant 50 particles. The expected behaviour of pressure inversely proportional to volume, as given by Boyle's law[10], can be seen in figure 12. The graph

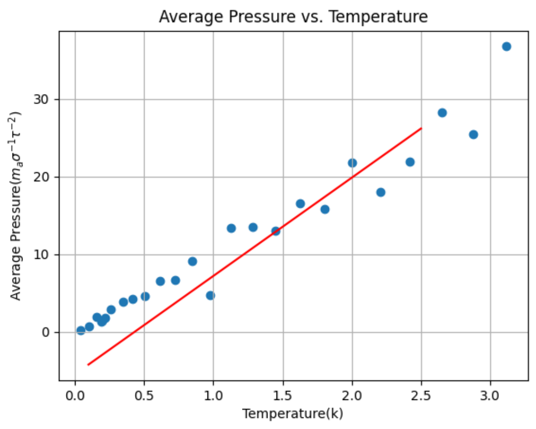


FIG. 11: Pressure dependence on temperature.

shows more of a deviation from a linear fit than figures 10 and 11. This is likely due to the increased box size reducing the number of particles counted for the pressure readings. A larger total time of simulation would likely rectify this, however with the data in figure taking over 30 minutes to collect it is outside of the scope of this paper to test this.

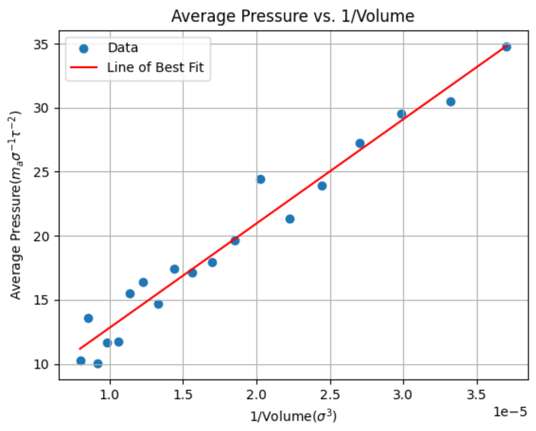


FIG. 12: The dependence of pressure on the volume of the system.

DISCUSSION

The method used in this simulation of evaluating the force on every particle from every other particle is known as direct integration or Cowell's method[8]. This is very computationally demanding and scales like $O(N^2)$, meaning that the number of computations to advance a single time step scales quadratically with the number of particles in the system.

This simulation could greatly benefit from a performance improvement such as that offered by a hierarchical tree algorithm. This involves the grouping of particles, enabling efficient computations by only considering interactions between particles from neighbouring cells individually. For particles located in distant cells, their effects can be approximated collectively. Treating the particles as a single large particle centred at the distant cell's centre of mass or through a low-order multipole expansion. This strategy significantly reduces computational complexity to $O(NlogN)$. This method reduces the accuracy but allows for higher-resolution simulations in feasible time frames.

CONCLUSION

Overall, the code succeeds in modelling the n-body problem using the Lennard-Jones potential for low N systems as evident from the various tests. The code in its current state serves as a good tool for building an intuition for n-body systems, likely useful as a supplementary resource for those manipulating the Lennard-Jones potential equation in problems of relatively low N. There is room for improvement in the code, particularly if it were to be used as a tool to model the behaviour of real-life systems with an N value to the order of Avogadro's constant. The biggest area for improvement is the integration method. As discussed, using a hierarchical tree method could offer a slightly less accurate but more efficient code.

REFERENCES

[1] Newton, I. (1967) Philosophiae Naturalis Principia Mathematica. London.

[2] Jones, J.E. (1924) 'On the determination of molecular fields.', Proc. Roy. Soc. , A 106, pp. 441–718. doi: `https://doi.org/10.1098/rspa.1924.0081.`.

[3] Schrödinger, E. (1926) 'Quantisierung als Eigenwertproblem', Annalen der Physik, 385(13), pp. 437–490. doi:10.1002/andp.19263851302.

[4] London, F., 1930. Über einige eigenschaften und anwendungen der molekularkräfte. Z. Phys. Chem. Abt, pp.222-251.

[5] Lennard-Jones, J.E. (1931) 'Cohesion', Proceedings of the Physical Society, 43(5), pp. 461–482. doi:10.1088/0959-5309/43/5/301.

[6] Verlet, L. (1967) 'Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules', Physical Review, 159(1), pp. 98–103. doi:10.1103/physrev.159.98.

[7] Barnes, J. and Hut, P. (1986) 'A hierarchical o(n log n) force-calculation algorithm', Nature, 324(6096), pp. 446–449. doi:10.1038/324446a0.

[8] Cowell, P.H. and Crommelin, A.C. (1910) 'The orbit elements of Halley's Comet', Astronomische Nachrichten, 185(17), pp. 265–268. doi:10.1002/asna.19101851704.

[9] Tippens, Paul E. (2007). Physics, 7th ed. McGraw-Hill. 386–387

[10] Levine, I.N. (1978) Physical Chemistry. New York: McGraw-Hill.