**3-D Molecular Dynamics of Argon Gas**

Brandon Ewert, Jacob Calcutt (Michigan State University), Selwyn Hanselmann (TU Delft)

**I - Introduction**

The dynamics of many body systems is nearly impossible to characterize without the help of a computer. This project aims to simulate the evolution of an inert gas from a low – energy configuration to thermodynamic equilibrium, using FORTRAN. We have chosen Argon gas in particular, since its lowest energy state is that of the convenient Face Centered Cubic (FCC) crystal lattice. Adding energy to the gas would provide kinetic energy that, over time, would allow atoms to dig out of their potential wells, thus breaking the symmetry of tightly knit intermolecular forces. As more energy disperses through the gas, the neatly paced lattice structure is broken up into fluid, flowing and interacting with surrounding atoms. Evolution of gas depends starkly on initial conditions and fundamental assumptions.

**II – Framework**

Evolution of gas depends starkly on initial conditions and fundamental assumptions. The fundamentals of our project are spelled out below:

1. Each atom will be interact with other atoms through the so-called Lennard-Jones Potential. This function gives the potential energy of an atom in terms of its separation distance from the other atom, r.

The constants ε and σ are intrinsic the gas. 4ε is the depth of the potential well – determined empirically for each gas, and σ is the separation distance of zero potential (effectively the size of the atom). The force experienced by the atom is the negative gradient of the potential, and can either point directly toward or away from the other atom, depending on its separation.

This potential includes a region of attraction, to simulate the attraction of temporary electric dipoles within the gas (Van der Waals / London Dispersion forces), as well as a much steeper repulsive term. The repulsion arises when two atoms come so close, the coulomb repulsion of their nuclei is no longer screened by the electrons orbiting the atom. Sandwiched between these regions of repulsion and attraction is critical point of the potential – the absolute minimum – corresponding to zero net force between two atoms.



**Figure 1**. A graph of the Lennard-Jones Potential and its derivative. Note that the position of the minimum of potential corresponds to zero force.

With a low kinetic energy, the atom oscillates in its potential well (Think water sloshing in a bucket). If enough energy is added, the atom may escape the well of the other, and be free to roam and interact with other atoms (You swung the bucket too hard!) In our simulation, many atoms will be involved, and the forces between atoms must be tracked constantly. The force between atoms will be crucial for calculating all other physical quantities (momenta, positions, pressure) as the gas evolves.

1. Atoms will initially be locked in an FCC lattice. Simply put, the entropy, or formal measure of disorder of a system, will always tend to increase over time. If we suspect that our gas may be able to form a different type of lattice after equilibrating, we should choose our initial conditions to be conducive to forming such a state. If we want to observe our gas evolve to a low-energy equilibrated state, it is best to begin by setting our gas in a lower-energy state than the one we intend to find. In this way, the entropy can increase all it wants while the FCC lattice breaks up, but there will not be so much disorder that the gas never finds its way to a low-energy equilibrium.
2. Atoms will be assigned random momenta within their lattice to kickstart their evolution. In a repeating lattice structure, there is no asymmetry to allow for interatomic forces to induce motion. The uniformity of the lattice means that if the gas particles have zero momentum, they will forever be locked in their lattice structure. Rather than place atoms randomly, we chose to assign random initial momenta so that the atoms may escape their potential wells and break up the lattice. Typical gas samples have momenta following a Gaussian distribution, whose standard deviation is determined by the temperature of the gas. The higher the temperature of the gas, the higher average momentum of each atom. The Gaussian distribution will be centered around momentum equal to zero. Otherwise, the gas would have some net momentum in some direction, causing the gas to converge or diverge wildly from its initial lattice.
3. All quantities, fundamental constants included, will be unitless. For the sake of calculation, it is simpler to write the Boltzmann constant as 1 rather than its true value in SI units. For this reason, and the fact that we are only looking for general trends – not exact physical predictions – we will set all fundamental constants to 1.
4. Time will evolve in quantized steps, not continuously. There are obvious limits on the resolution of our simulation imposed by computer processing speed. Initial momenta of gas particles will induce rapidly-changing forces, which will in turn affect the momenta and trajectories of atoms at a later time. In order to strike a balance between sampling forces once every hour, and once every Planck-second, we must find a convenient time step after which to calculate new potentials, forces, momenta, and positions. This time step that most efficiently enriches our resolution of how the atoms evolve, but still keeps our rate of calculation reasonable for computers is .004 (unitless, of course).
5. Momentum and Positon will be updated after each time step as prescribed by a simple algorithm. The algorithm is based off the fact that integrating force with respect to time will yield momentum, and integrating once more with respect to time will yield position (after cancelling out mass, of course). Its exact recipe is described in section III.
6. We will keep the temperature of our gas constant during evolution. Temperature, which is proportional to the average kinetic energy of the atoms in our gas, will be fixed at a specific value. During the first stages of the gas’s evolution, its kinetic energy will vary drastically as pent-up potential energy in the lattice bonds converts to kinetic energy. In order to keep our gas from any anomalistic jumps in energy, we can sample energy from average kinetic energy, and renormalize the temperature value until it equilibrates to a reasonable value. Once this has been done over several time steps, the gas should evolve normally.
7. During the evolution of our gas, we will calculate pressure and pair correlation. We have predictions about what macroscopic pressure of a gas will be. This project will allow us to mesh macroscopic observation with our simulated mesoscopic system. Less commonly mentioned pair correlation will be another quantity that is useful to measure. It allows us to probe the coherence of our gas, and watch it evolve from perfect lattice (ice) to fluid, and will accentuate any pattern in the equilibrated gas’s spatial distribution.
8. We will simulate our gas as infinite in spatial expanse. Employing periodic boundary conditions will effectively remove any limits imposed by physical barriers to the gas. Rather than observing the gas effuse from a compact solid to fill a physical container, we observe a small portion of a much larger system of gas. Periodic boundaries ensure that whatever interactions are seen on one edge of our cubic “boundary” affects atoms on the opposite “boundary”. I use quotations because our “boundary” is much more akin to a window into a sample of gas of much larger extent. As we see atoms escape one side of the window, new atoms appear on the other side to fill the void. We chose our boundary window to be a cube, for simplicity.

**III – Code**

We begin modeling our gas by first developing the layout of the lattice. Since we know our gas condenses in an FCC lattice, we want to find the most convenient way to tessellate a unit cell into a many-cell three dimensional lattice. The canonical unit cell is cubic, and contains four whole atoms. Figure 2 depicts the unit cell as a composite of 14 separate atoms. The unit cell only occupies the shaded region, however. If we imagine slicing off the parts of atoms that extend outside the range of the box, we would be left with six half-atoms, and eight eighth-atoms, totaling four whole atoms. This particular choice of unit cell shape is arbitrary, just as long as the number of atoms that comprise it remains four. Naively, one may think a cubic unit cell would be ideal for a cubic window. Not so! Partial-atoms of the cubic cell would make meshing opposite boundaries with one another needlessly complicated. For this reason, we chose a simpler arrangement of four atoms that will form the framework of our initial lattice. As shown in Figure 3, we chose 4 whole atoms – one corner and three centers. This is the smallest collection of gas particles we can make. If we want more, we have to add perfect cube multiples of these cells, and arrange them so that the corner atom of each cell is spaced by the length of a cell. The next smallest collection would be eight (two cubed), and each corner atom of the unit cell would be situated at the corner of the cube in Figure 3. Any larger collection (27 cells, 64, 125, and so on) would just build off of our cubic pattern. In this way, there will be corrugated edges that mesh seamlessly with the opposite edge. This will assure all atoms experience no initial force, and their motion will stem only from their randomly assigned momenta.

**Figure 2**. An FCC unit cell. Note that all atoms extend beyond the boundary of the cell.

To construct this, we iterate along the x dimension, placing bases up to the dimension of the lattice, then increment one step in the y, and lay down another set of bases in the x. Once a level in the y axis has been filled, the pattern starts over for each level in the z axis. All of this is contained within the subroutine *position\_initializer*. The function passes in an empty array called position, as well as the dimension of the lattice. The array contains all positions components (x,y,z) of the atoms, and is returned so that our other functions will later update it as the positions change. Since position is a vector, the array has 3 rows for each dimension, and a number of columns equal to the number of particles. The aforementioned step length is dubbed our ‘cell\_width’ within the subroutine.

**Figure 3**. Our choice of basis.

To assign momenta, we must first find a random momentum. To do this, we construct a Gaussian random number generator with a constantly changing seed (time is ideal). Subroutine *get\_gaussian\_momentum* does just as its name suggests – it grabs random floats between zero and one, and compares their value to the expected Gaussian distribution of the gas’s momentum. If the random number is within the range of the exponential, is keeps the value, multiplies by the atom’s mass (1!), and adds the momentum value to an array of momenta. This three dimensional array, like the position array, will be returned and updated later as the gas evolves.

Our program initializes three other arrays – one for potential energy, kinetic energy, and force. The force array is 3 dimensional, but since energy is a scalar, there is only one row for those arrays. Force is initialized at zero, and both energies are calculated in the *calc\_energy* subroutine. Kinetic energy is calculated from the momentum (proportional to the velocity through the mass, which is equal to one), and potential is calculated using the Lennard-Jones formula.

To simplify our calculations, however, we ignore contributions to the potential energy beyond a maximum separation distance. The optimal radius of influence was determined empirically to be 3.2. This way, we can have sufficiently complicated interactions to flesh out intricate details, but save from calculating the interaction between *every* particle. In calculating separation distances, we take the difference of two position vectors. Since the boundaries are periodic, there are two possible values, though, as shown in Figure 4. We choose the shorter of the two as their “true” separation, by taking the modulo of the distance and half of the window length.

Once the force has been determined, we begin updating our quantities. The impulse – momentum theorem tell us that we can approximate forces over small time intervals as roughly constant. The product of the force and the time step will yield the change in momentum, and can be added to the current momentum. For an even better approximation, we add half of the expected impulse, update the position, recalculate the force, and add a second half-step impulse. Figure 5 shows us this process more explicitly. This is essentially the core of our code. Force induces a change in velocity and position, which is used to calculate a new force. Our subroutine *update* contains just one more line – one for updating kinetic energy. We keep track of this quantity because not only is it necessary to calculate the temperature of the gas, it also lets us track the conservation of energy in the system. Once equilibrated, the kinetic energy will remain fairly constant. Any drastic fluctuations would be indicative of a bug or two.

**Figure 5**. Our update algorithm. delta\_t is our time step, and velocity is really our momentum array with mass for each atom.

velocity = velocity + force\*delta\_t\*.5

position = modulo((position + velocity\*delta\_t),length)

call calc\_force(position, force, length, N,temp\_in,pressure)

velocity = velocity + force\*delta\_t\*.5

**Figure 4**. The atoms’ actual x and y separation coordinates are given as dashed lines. The arrows represent the alternative “Pac Man”-style separation distance.

To monitor total pressure, we must combine the intrinsic pressure carried by the gas in the form of kinetic energy, and combine it with the pressure induced by interatomic forces. The formula we use is known as the *virial equation*:

Here, N is the number of atoms, V is the volume (length of the window, cubed), τ is the temperature (we set the Boltzmann constant to one), and the summation is of the magnitude of the dot product of the force experienced by a particle with its separation distance, over all particles. This gives a comprehensive and realistic estimate of the pressure of the gas as a whole. We calculate this pressure on each call of the *calc\_force* function.

To facilitate a proper evolution of the gas, we maintain the temperature at a constant value for a small fraction of our total number of steps. We do this by multiplying our entire momentum array by a factor of the square root of our desired temperature divided by our current temperature. Over these active steps (About 40 out of 1000 total), the temperature is in rapid fluctuation due to high initial forces. Our desired, or ‘hold’ temperature typically multiplies momenta by a factor much greater than one, as we will see in our results section. This is entirely dependent on the arbitrary choice of holding temperature and height of the Gaussian of the momentum distribution.

Our final calculated value, pair correlation, is calculated only once out of our entire set of iterations. We chose to look only at its plot after the system has evolved completely, to see the smoothest characterization. Pair correlation, given by the curious formula below, is a measure of a gas’s relative spatial distribution. A histogram is included in our results section.

This function approximates the number of particles at a given separation distance. The shape of the distribution is intrinsic to the form of the intermolecular force.

**IV – Results**

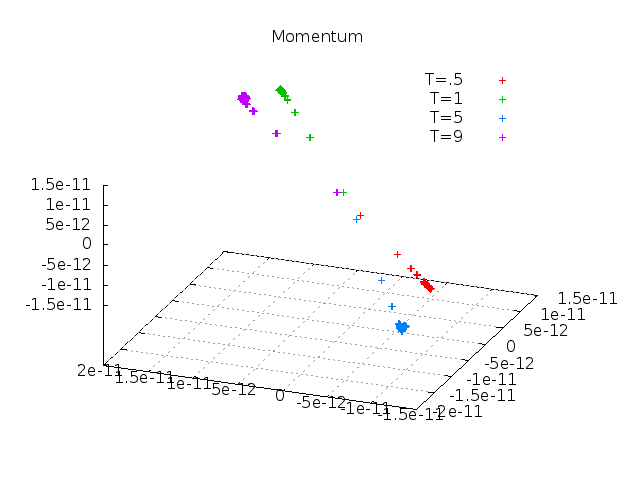
After rigorously testing our code, we have compiled a plots of our relevant quantities. We used wrote our data to external files, from which GNUPLOT interpreted and plotted the results. All plots are over a period of 1000 time steps, with 500 atoms (dimension 5). Without further ado:

**Momentum (Figure 6.)**

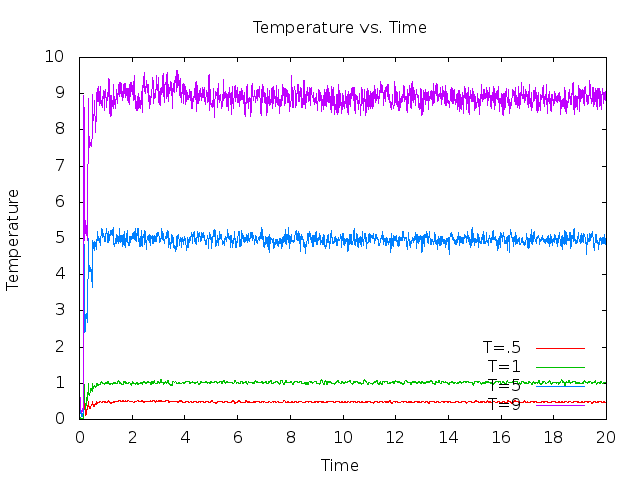
This plot shows the three-dimensional average momentum of all particles, with one point plotted per time step. We ran our program at four different holding temperatures to simulate Argon’s evolution at a solid, liquid, and gaseous state. The higher the temperature, the higher the average momentum. The plot shows a densely packed region of momentum (conservation after holding temperature has been reached), and a slight tail that extends almost to (0, 0, 0). This tail represents the rise/descent from initial momentum to a stable equilibrated average. Not much surprised us here – momentum stays relatively constant, corresponding to almost no energy loss.

**Temperature (Figure 7.)**

We’ve plotted four different temperatures to see how temperature fluctuates over time. Apparently, at higher temperatures, we see a higher amplitude in our fluctuation, possibly due to the higher momenta, and thus higher chance for atoms to come closer together and experience higher forces. These forces then induce higher average kinetic energy and, finally, a higher temperature.



**Figure 6**. Vector Momentum at selected temperatures.

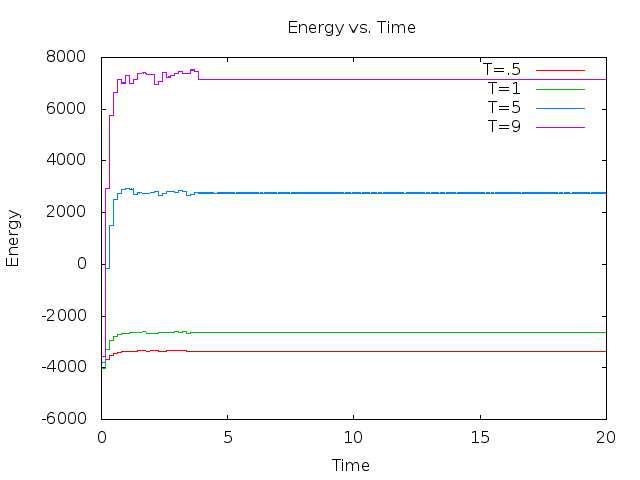


**Figure 7**. Temperature fluctuations over time.

**Energy (Figure 8.)**

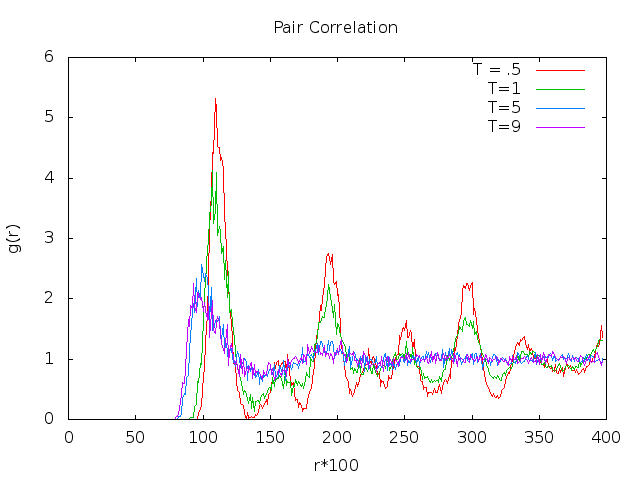
As any closed system should, we want to make sure our gas’s total energy remains constant. Figure 8 shows the total energy plotted versus time. As expected, the first unit of time is unstable from the holding temperature factor. What is interesting, however, is the period directly after the holding temperature is reached – the energy fluctuates with a sizable amplitude for nearly 20% of the total evolution time. After this period, the energy is conserved to an uncertainty of around 1% of the total energy for each temperature. Note that initial total energy is negative, since all atoms begin with bound energy from their lattice interactions. It is safe to say that our truncation of our force calculation is acceptable. We see no energy loss over time due to our disposal of long-range force interactions.

**Figure 8.** Energy over time. Flat lines make me smile!



**Pair Correlation (Figure 9)**

Our pair correlation is consistent with Lennard-Jones predictions, which are remarkable intricate. We see oscillations around our normalized value of one for g(r). At higher temperatures, the initial peak creeps slowly to the left, as we would expect. As discussed before, the higher temperatures allow atoms to get closer to one another. The amplitude of the damped oscillations decrease with higher temperature, likely due to kinetic energy dominating over potential energy. Note that g(r) is zero in the region occupied by the atom itself (r ~ 1) for low temperatures, but our classical assumption of solid atoms breaks down at high temperatures. As its graph suggests, there is a finite slope of V(r) at r = σ, meaning the atom is not impenetrable by other atoms. Though the force is extraordinary, high temperature grants atoms with high momenta, turning our gas from a sea of marbles into a sea of foam balls.



**Figure 9**. Pair Correlation. Note that relative positions of peaks are dependent on temperature.

**V – Summary**

Our code adequately describes the evolution of Argon gas. Momentum and energy are conserved within the gas, as they only interact with themselves, and our truncation of force calculation is reasonable. Our temperature, once manually stabilized, stays constant. The pair correlation of our gas atoms is as expected, and its curves are mesmerizing. Our code, though not the fastest, runs nicely and produces neat plots.

**VI – Conclusion**

We all learning much about molecular dynamics through this project, and perhaps even more about proper coding techniques. Any redundancy or lack of optimization can drastically increase the time the program needs to run. We’ve learned that the hard way. We also know not to take subroutines with 27 input variables seriously. All in all, we could have been much more quantitative with our results, but general trends are what’s important. In further refinement, we could perform real-time visualizations of the gas in flux, and measure more involved quantities like free energy, entropy, and fugacity (Yes! I finally got to use that word in a sentence!)