Molecular dynamics model of solid-liquid phase transition in argon

Timofey Golubev

May 2, 2017

Abstract

An object-oriented molecular dynamics (MD) code was developed and used to model the melting of argon. The simulation is initialized with 500 atoms placed into a face-centered cubic structure with the atoms given initial velocities based on the Maxwell-Boltzmann distribution corresponding to a chosen temperature. The interatomic interactions are modeled by the Lennard-Jones potential. Then the system is allowed to evolve by solving classical equations of motion with the velocity Verlet method. The time-evolution of the atomic motions at different temperatures is studied. The system's energies, temperature, and diffusion coefficient are calculated and analyzed to find the crystal's melting point. It is found that the calculated melting point has some variation with different simulation approaches. The upper limit of the melting point is determined to be about 290K.

Contents

1 Introduction **2**

2 Theoretical background **3**
2.1 Classical equations of motion 3
2.2 Crystal structure 4
2.3 Interatomic forces 4
2.4 Equipartition theorem and temperature 6
2.5 Diffusion 6

3 The MD algorithm **6**
3.1 Initial Conditions 7
3.2 Boundary conditions 7
3.3 Interaction model 9
3.4 Integrator 9
3.5 Statistical Ensembles 11
3.6 MD Units 12
3.7 Visualization 13
3.8 Program flow 13
3.9 CPU Time 14

4 Results and discussion **14**
4.1 NVE Ensemble 14
4.2 Gradual heating 16
4.3 NVT Ensemble 19
4.4 Comparison with literature 20

5 Conclusion **22**

References **22**

Appendix **23**

1 Introduction

Molecular dynamics simulations are widely used in physics, chemistry, and biology as a bridge between theory and experiment. They allow to both qualitatively and quantitatively test theoretical models for the interactions and various processes in a system. For anything, but the smallest systems, it would be extremely tedious or impossible to solve all of the equations by hand. The computational results can also be used to improve our understanding of experimental results or even to predict certain processes and guide experiments. In addition, the computational physicist is not limited by the laboratory equipment, which in the personal experience of the author, has a tendency to not function as desired.

The first molecular dynamics simulation using a continuous potential was a study of argon by A. Rahman in 1964[1]. Being a noble element, the system of argon is relatively simple and therefore is a good system to use in order to describe and demonstrate the implementation of molecular dynamics. More specifically, I focus on the solid to liquid phase transition of argon and use several approaches to determine its melting point.

This report is organized as follows. Section 2 discusses the theoretical background of the molecular dynamics method, relevant solid-state physics, and how statistical physics can be used to calculate system properties. Section 3 discusses the main aspects of a molecular dynamics program and how these were implemented for the study of argon. Section 4 includes a discussion of the results for the different simulation approaches used and a comparison with literature. Conclusions are drawn in Section 5.

2 Theoretical background

Molecular dynamics makes use of classical equations of motion to study atomic and molecular systems and their properties. The basic concept is to chose a mathematical model for the interactions between system constituents and solve the equations of motions to generate particle trajectories. Then many system properties such as its structure, equation of state, transport coefficients, and non-equilibrium response (i.e. due to deformation) can be studied.

2.1 Classical equations of motion

For an N particle system, motion of each particle i is described by

$$\mathbf{f_i} = m_i \ddot{\mathbf{r}}_\mathbf{i} = -\frac{\partial}{\partial \mathbf{r_i}} \mathcal{U}(\mathbf{r_1}, \mathbf{r_2}, ..., \mathbf{r_N}) \tag{1}$$

where $\mathcal{U}$ is the inter-particle potential and is a function of the particle positions. When using these classical equations to model processes on the atomic scale, two main assumptions are made. One is the Born-Oppenheimer approximation which states that the electron dynamics are so much faster than atomic nuclei motions that the electrons can be considered to react instantaneously to their nucleus' motion. This allows to ignore electron motions in the model. The other assumption is that the nucleus motion is far from the Heisenberg uncertainty regime. That is $\Delta E \Delta t >> \hbar/2$. Using $\Delta E \sim k_b T$ and $\Delta t = 1/\omega$ where ω is the frequency of the atomic vibrations, this means that $T >> \hbar\omega/k_b$ must be satisfied. In solid state physics, the temperature of a crystal's highest normal mode of vibration is given by $T_{Debye} = \hbar\omega_{max}/k_b$, therefore $T >> T_{Debye}$ to be able to safely use classical physics (without applying corrections) for predicting physical quantities such as heat capacities. This is actually a quite strict requirement since Debye temperature is not necessarily low. For example, silicon has $T_{Debye} = 645K$ and argon has $T_{Debye} = 92K$ [2]. If this condition is not satisfied, the calculated thermodynamic quantities such as heat capacities could have a discrepancy with the true value.

2.2 Crystal structure

For molecular dynamics study of crystalline materials, the simulation is often initialized by placing atoms into an appropriate crystal structure. It is convenient and convention to describe crystal structures by their unit cells where the entire lattice can be created by repeating the unit cells. The origin of each unit cell is given by $R_{i,j,k} = a(i\hat{\mathbf{x}} + j\hat{\mathbf{y}} + k\hat{\mathbf{z}})$ where i, j, k enumerate the cells and a is the side length of each unit cell and is called the lattice constant. The atomic positions within each unit cell are given by the primitive lattice vectors. Solid argon forms a face-centered cubic (fcc) lattice which can be described by primitive lattice vectors

$$\begin{aligned}
\mathbf{r}_1 &= 0\hat{\mathbf{x}} + 0\hat{\mathbf{y}} + 0\hat{\mathbf{z}}, \\
\mathbf{r}_2 &= \frac{a}{2}\hat{\mathbf{x}} + \frac{a}{2}\hat{\mathbf{y}} + 0\hat{\mathbf{z}}, \\
\mathbf{r}_3 &= 0\hat{\mathbf{x}} + \frac{a}{2}\hat{\mathbf{y}} + \frac{a}{2}\hat{\mathbf{z}}, \\
\mathbf{r}_4 &= \frac{a}{2}\hat{\mathbf{x}} + 0\hat{\mathbf{y}} + \frac{a}{2}\hat{\mathbf{z}}.
\end{aligned}$$

Argon has a lattice constant of $a = 5.256$ angstrom [4].

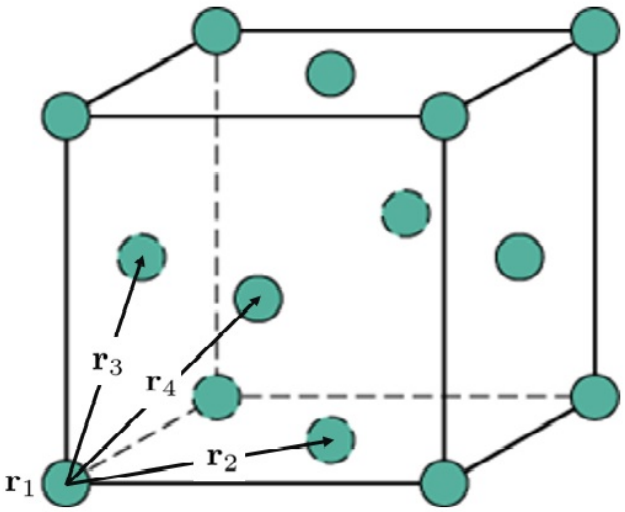

Figure 1: Face-centered cubic lattice with the primitive lattice vectors labeled. Adapted from [3].

2.3 Interatomic forces

A proper description of interatomic forces is essential for an accurate MD model. Typically, only pair-wise interactions are considered. Most commonly used pair

potentials for systems without electrostatic interactions have a similar structure. The finite diameter of the particles is modeled by a steep repulsive potential such as an exponential or $1/r^n$ where $n \geq 9$. There is also an attractive $1/r^6$ potential due to London dispersion forces which arise when instantaneous created dipoles induce dipoles on neighboring particles. The shape of these potentials is shown in Figure 2.

For simple systems such as noble gases which have a completely filled valence electron shell, the so-called *hard-core* potentials are a good model. The most well-known is the Lennard-Jones potential

$$U(r_{ij}) = 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right], \tag{2}$$

where $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$ is the distance from atom i to atom j, ϵ determines the depth of the potential, and σ is the diameter of each atom and also the distance at which potential is zero. The force exerted on the ith atom due to the jth atom can be calculated by $\mathbf{F}_{ij} = -\nabla U(r_{ij})$ yielding

$$\mathbf{F}_{ij} = 24\epsilon \left(\frac{\sigma}{r_{ij}^2} \right) \left[2 \left(\frac{\sigma}{r_{ij}} \right)^{11} - \left(\frac{\sigma}{r_{ij}} \right)^{5} \right] \left(\frac{x_{ij}\hat{\mathbf{x}}}{r_{ij}} + \frac{y_{ij}\hat{\mathbf{y}}}{r_{ij}} + \frac{z_{ij}\hat{\mathbf{z}}}{r_{ij}} \right) \tag{3}$$

where $x_{ij} = x_i - x_j$ is the x-component of displacement between the two particles. The other components of displacement y_{ij} and z_{ij} are defined in the same way.

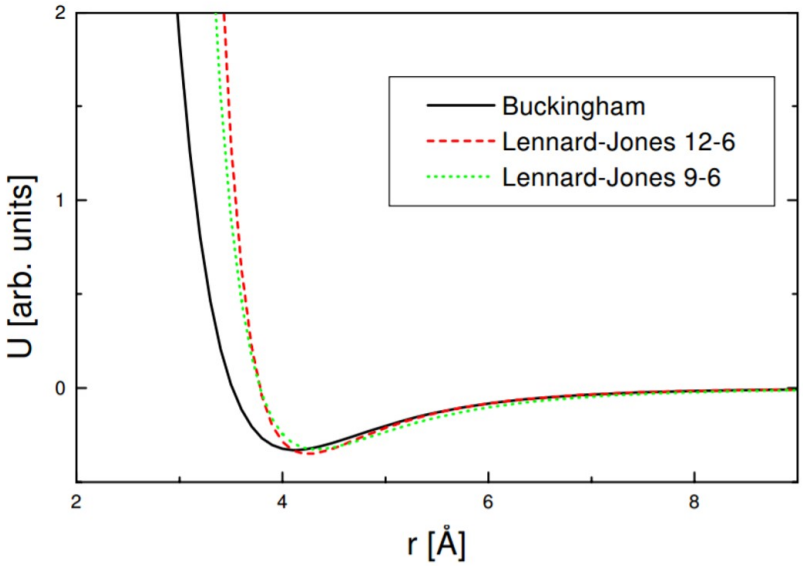

Figure 2: Some commonly used pairwise potentials [5].

For some systems, the $1/r^{12}$ repulsive term may be too steep which for example could cause an overestimation of the pressure. In these systems, a less steep $1/r^9$ term may be used (Lennard-Jones 9-6 in Figure 2), or it may be replaced by an exponential in which case this is called a *soft-core* potential (two examples are the Buckingham and Morse potentials) [5].

2.4 Equipartition theorem and temperature

From statistical physics, we know that when a system is in thermal equilibrium, every degree of freedom which appears quadratically in the expression for energy contributes $1/2k_BT$ to the system's average energy. Let's consider the contribution to a system's kinetic energy which is

$$E_k = \sum_{i=1}^{N_{\text{atoms}}} \frac{1}{2} m_i v_i^2 \tag{4}$$

where $v_i^2 = v_{ix}^2 + v_{iy}^2 + v_{iz}^2$. Therefore, there are 3N squared terms in this expression for kinetic energy so according to the equipartition theorem the average kinetic energy is

$$\langle E_k \rangle = \frac{3}{2} N_{\text{atoms}} k_B T \tag{5}$$

We can use this to approximate the system's instantaneous temperature corresponding to a certain kinetic energy (calculated by equation 4)

$$T = \frac{2}{3} \frac{E_k}{N_{\text{atoms}} k_B} \tag{6}$$

2.5 Diffusion

The Einstein relation relates mean square displacement $r_i^2(t) = |\mathbf{r}_i(t) - \mathbf{r}_i(0)|^2$ to the diffusion coefficient D and time t

$$\langle r^2(t) \rangle = 6Dt \tag{7}$$

Therefore, we can calculate the diffusion coefficient by using collected data about the mean square displacement of atoms at a given time [6]

3 The MD algorithm

The three main ingredients of a molecular dynamics simulation are an interaction model, an integrator, and a statistical ensemble. The interaction model describes the forces between the particles, the integrator advances the system in time, and the ensemble determines which of the system properties will be held constant. One must also consider how to initialize the system, treat the system boundaries, and choose a convenient system of units. Once the simulation is complete, often a visualization tool to view the trajectories of the particles is used.

3.1 Initial Conditions

The atoms were initially placed in an fcc lattice (see Section 2.2) and given velocities corresponding to a chosen system temperature T according to the Maxwell-Boltzmann probability distribution

$$P(v_i)d\mathrm{v}_i = \left(\frac{m}{2\pi k_B T}\right)^{1/2} \exp\left(-\frac{mv_i^2}{2k_B T}\right) d\mathrm{v}_i \tag{8}$$

$P(v_i)d\mathrm{v}_i$ is the probability of finding a particle having velocity in the range between v_i and $v_i + dv_i$. This distribution describes the velocities of atoms in an ideal gas in thermal equilibrium. It does not take into account interactions between particles, but for systems of ideal gas like argon, these interactions are small.

3.2 Boundary conditions

Since a computer simulation is always of finite size, one must specify how the boundaries are treated. There are several commonly used boundary conditions. For studies of individual molecules or small clusters, free boundary conditions (FBCs) work well. In FBCs, there are essentially no boundaries, the system constituents are allowed to reach any positions. For simulating larger systems (i.e bulk crystals), periodic boundary conditions (PBCs) are best. Here it is imagined that there are infinite repeating versions of the simulation cell surrounding it on all sides (Figure 3). These conditions allow one to study very large systems by breaking them down into small periodic units. For some systems, for example study of 2D or 1D materials, mixed boundary conditions may be best. In this case, there are PBCs in some directions and FBCs in others. Periodic boundary conditions were used for this study except for some qualitative experiments done with FBCs and mixed boundary conditions.

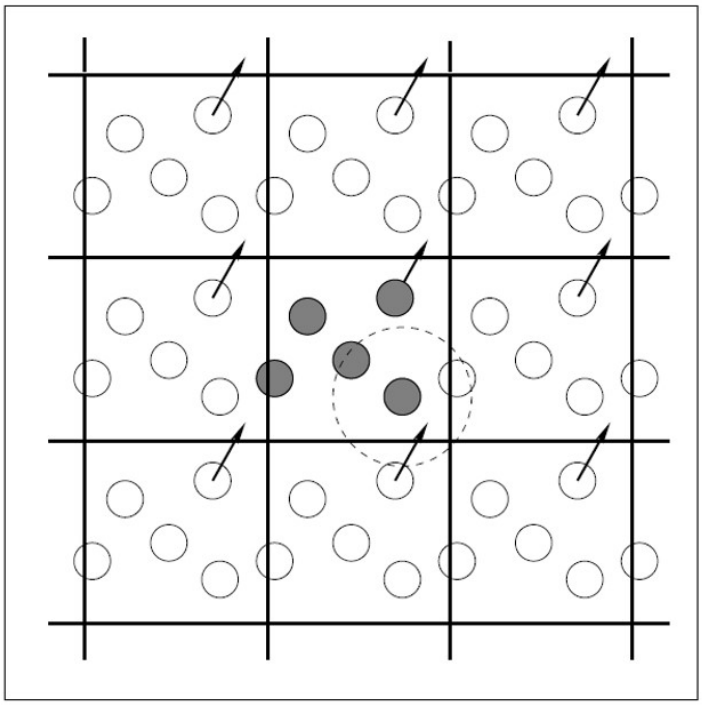

Figure 3: Periodic boundary conditions [8]

PBCs and minimum image convention

In PBCs we explicitly keep track of only atoms within the simulation cell. When particles cross a boundary of the simulation cell, they emerge on the other side. In this way, the number of particles in the cell is conserved and the effects of surfaces are removed. If the inter-particle interactions can be described by a short range potential (i.e. potentials of the Lennard-Jones type), then the minimum image convention may be used. In this convention, each particle interacts with each other particle or its image, whichever is closer. More specifically, if the displacement between two interacting particles in any dimension is greater than half of the simulation cell side length, then the image (in the neighboring imaginary cell) of the other particle will be inevitably closer and the interaction with the image will therefore be used. This convention ensures that the same sphere of interaction range is used for all particles in the cell, even if they are near the cell edge[7].

The periodic boundary conditions are enforced at every timestep by calling the function shown below.

```
void System::applyPeriodicBoundaryConditions() {
  for(Atom *atom : atoms()) {
    for(int j=0;j<3;j++){
```

```
      if (atom->position[j] <  0.){
        atom->position[j] += m_systemSize[j];
      }
      if (atom->position[j] > m_systemSize[j]){
        atom->position[j] -= m_systemSize[j];
      }
    }
  }
}
```

The minimum image convention is applied when calculated the displacements between atom pairs in the `LennardJones` class. These displacements are then used to calculated the forces.

```
vec3 displacement;
for(int j=0;j<3;j++){
  displacement[j] = current_atom->position[j]
  -other_atom->position[j];
  if(displacement[j] >  system.systemSize(j) * 0.5){
    displacement[j] -= system.systemSize(j);
  }
  if(displacement[j] <= -system.systemSize(j) * 0.5){
    displacement[j] += system.systemSize(j);
  }
}
```

3.3 Interaction model

The calculation of the forces is the most computationally intensive part of a MD program since to calculate the force on a single atom requires computing the contribution from all other atoms (or images) within the interaction range. Also this force calculation must be repeated for each timestep. For this study, the Lennard Jones force is used. Repeatedly used quantities are precalculated and the formula was simplified algebraically in order to save floating point operations. The CPU time is halved by using Newton's third law $Fij = -F_{ji}$ and only calculating the force once per pair. The source code is given in the Appendix.

3.4 Integrator

The integrator is the algorithm which numerically integrates the differential equations of motion. For a good MD code, there are four main criteria an integrator must satisfy [5]:

- Accurate: approximate true trajectory well (can test with simple systems for which have analytic solutions)
- Stable: conserve energy and momentum
- Robust: allow for large enough time steps in order to propagate system efficiently
- Efficient: only use 1 force evaluation per time-step

The selection of an appropriate timestep is very important. The time step must be on the order of or smaller than the fastest motions of the system which is typically $O(10^{-15}\text{s})$. Of course, the kinetics of the system will increase with temperature, therefore smaller timesteps are necessary for studying systems at higher temperature.

velocity Verlet

The velocity Verlet integrator satisfies all above criteria and is widely used for MD simulations. The method[9] is based on the Taylor expansion of both the velocity and the displacement

$$\begin{aligned} x_{i+1} &= x_i + hx'_i + x''_i\frac{h^2}{2} + O(h^3) \\ v_{i+1} &= v_i + hv'_i + v''_i\frac{h^2}{2} + O(h^3) \end{aligned} \tag{9}$$

If we consider again the Taylor expansion $v'_{i+1} = v'_i + v''_i h + O(h)$, we will have the $h^2 v'' = h(v'_{i+1} - v'_i) + O(h^3)$. If we plug this relation into equation (9), we will have:

$$\begin{aligned} v_{i+1} &= v_i + hv'_i + \frac{h}{2}(v'_{i+1} - v'_i) + O(h^3) \\ &= v_i + \frac{h}{2}(v'_{i+1} + v'_i) + O(h^3) \end{aligned} \tag{10}$$

Using the fact that the derivative of velocity v' is the acceleration a, we have the velocity Verlet method:

$$\begin{aligned} x_{i+1} &= x_i + hv_i + \frac{h^2}{2}a_i + O(h^3) \\ v_{i+1} &= v_i + \frac{h}{2}(a_{i+1} + a_i) + O(h^3) \end{aligned} \tag{11}$$

If we have the position and velocity of the first point, we will be able to calculate the velocity and position of next point.

Code implementation:

The method was implemented in the `VelocityVerlet` class as follows. Note that by reusing the forces found from the previous iteration of the loop, we only have one force calculation per time-step even though the method requires two accelerations (a_i and a_{i+1}) for each step.

```
if(m_firstStep) {
  system.calculateForces();
  m_firstStep = false;
}

for(Atom *atom : system.atoms()) {
  atom->velocity += atom->force*0.5*dt/atom->mass(); //ith velocity
  atom->position += atom->velocity*dt;  //i+1th position
}
```

```
system.applyPeriodicBoundaryConditions();
system.calculateForces(); //recompute forces
//calculate new velocities (i+1th)
for(Atom *atom : system.atoms()) {
  atom->velocity += atom->force*0.5*dt/atom->mass();
}
```

3.5 Statistical Ensembles

Statistical ensembles used in MD simulations are the same as in statistical physics. Each ensemble keeps certain physical quantities constant. However, since in computer simulations there is error in the integrator scheme and round-off errors from limited precision of number representation, these quantities will only approximately be kept constant. The microcanonical ensemble (aka NVE) keeps the particle number N, system volume V, and system total energy E constant. This is the natural ensemble for MD since classical equations of motion of a fixed size system naturally conserve energy. However, for comparison with experimental data, it is often useful to use the canonical ensemble (aka NVT) where the temperature T instead of the total energy is kept constant. For this study, I use both the NVE and NVT ensembles as well as a simulated gradual heating of the system.

Microcanonical

Being the natural ensemble for MD, NVE is the simplest to implement. The atom velocities are initialized according to the Maxwell-Boltzmann distribution corresponding to a certain user-selected temperature (see Section 3.1). Then the system is allowed to evolve naturally. The temperature is monitored but not adjusted.

Canonical

To use the NVT ensemble, some modifications must be made to the algorithm. The simplest method to keep temperature constant is to periodically rescale the particle velocities by using the equipartition theorem. We can calculate the system's instantaneous temperature by combining equations 4 and 6 and using the fact that for a monatomic system, $m_i = m$.

$$T = \frac{m \sum_{i=1}^{N_{\text{atoms}}} v_i^2}{3N_{\text{atoms}} k_B} \tag{12}$$

If we take the ratio of two temperatures

$$\frac{T_2}{T_1} = \sum_{i=1}^{N_{\text{atoms}}} v_{2i}^2 \Big/ \sum_{i=1}^{N_{\text{atoms}}} v_{1i}^2 \tag{13}$$

which means that

$$\sum_{i=1}^{N_{\text{atoms}}} v_{2i}^2 = \sum_{i=1}^{N_{\text{atoms}}} \frac{T_2}{T_1} v_{1i}^2 \tag{14}$$

So if we have a set of velocities v_{1i} which correspond to system temperature T_1, to increase the system's temperature to T_2 we can rescale each of the velocities

$$v_{2i} = \sqrt{\frac{T_2}{T_1}} v_{1i} \tag{15}$$

This scaling is implemented in the function below.

```
void System::rescaleVelocities(StatisticsSampler &statisticsSampler, double
     desiredTemperature){
 double rescaling_factor = sqrt(desiredTemperature/statisticsSampler.
      temperature()); //sqrt(T_desired/T_actual)

 for(Atom *atom : atoms()) {
   atom->velocity *= rescaling_factor;
 }
 removeTotalMomentum();  //to eliminate system drift
}
```

Gradual heating

If we consider how melting temperature would be studied experimentally, one would place the solid in contact with a heating source (i.e. a hotplate) which can be regulated and observe what happens. A good experimentalist would also measure the temperature of the solid itself with, for example, a thermocouple. The solid's temperature will gradually increase as it exchanges heat with the heating source. In order to model this situation, I can pick an initial simulation temperature at which I am sure that the material remains a solid. This would be equivalent to the ambient temperature of the object before the experimenter places it on the hot plate. Then, at every MD time step, I increase the system's temperature by a very small amount δT by rescaling the atomic velocities in the same way as I did for NVT, expect now $T_2 = T_1 + \delta T$

```
void System::increaseTemperature(StatisticsSampler &statisticsSampler, double
      increment){
 //increases system temperature by factor
 double velocity_rescaling_factor =
  sqrt((statisticsSampler.temperature()+increment)/statisticsSampler.
       temperature());
 for(Atom *atom : atoms()) {
  atom->velocity *= velocity_rescaling_factor;  //a*=b means a = a*b
 }
}
```

3.6 MD Units

Molecular dynamics programs often use a set of so-called MD units which make the calculations more convenient and reduce the amount of operations necessary. This program uses the following units [6].

- 1 unit of mass = 1 a.m.u = 1.661×10^{-27}kg,
- 1 unit of length = $\sigma = 3.405 \times 10^{-10}$m,
- 1 unit of energy = $\epsilon = 1.651 \times 10^{-21}$J,
- 1 unit of temperature = 119.735K.
- 1 unit of time = 3.41263×10^{-13}s

Note that I chose to use the diameter of argon (σ) as the unit for length in order to minimize the amount of computations in the Lennard Jones force and potential calculations.

3.7 Visualization

A very popular way to output the positions of the atoms over the course of time is to use an `.xyz` file. This is simply a specially formatted text file to which one outputs the position coordinates of all particles in the system at periodic time intervals. Then one can use a visualization software to view the motion of these particles and make animations. For this project, I use the free software *Ovito*[10].

3.8 Program flow

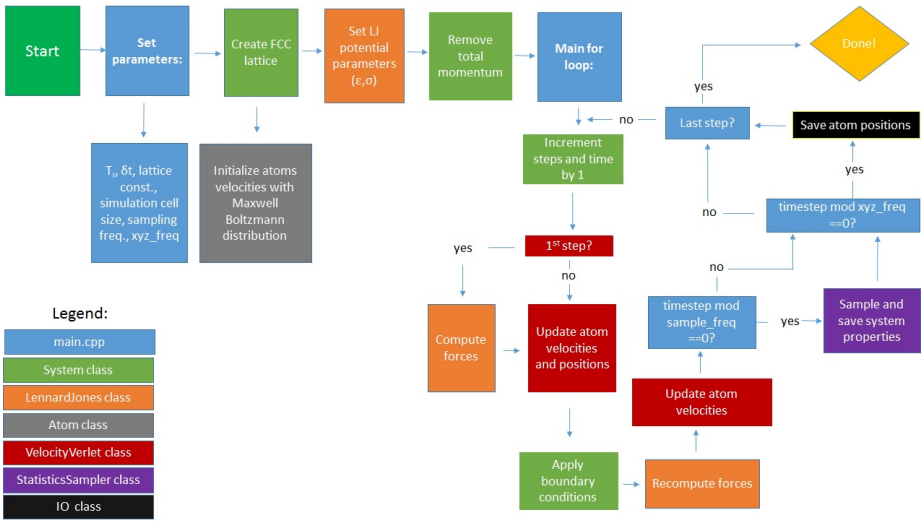

Figure 4: Flowchart of the program. Colors correspond the classes in which the various operations where performed and are defined in the legend.

3.9 CPU Time

Molecular dynamics simulations are computationally intensive and the CPU time increases dramatically with increasing system size. In my program, the CPU time to perform a given number of MD steps increases as approximately $O(N^2)$ where N is the number of atoms. There are methods such as using an interaction cut-off radius or neighbor lists [2] to reduce the CPU time, but I did not explore these in this initial study. The CPU times for performing 10,000 MD steps with different system sizes are given in the table below. The time increases linearly with the number of steps so the time necessary for longer runs can be estimated from this table. The most computationally intensive runs done in this study were with a system of 500 atoms and 2 million MD steps, taking approximately 6-7 hours to complete.

Number of atoms	CPU Time (s)
108	5.7
256	30.2
500	113.9
864	356.9
1372	929.8

Table 1: Computation time for different system sizes.

4 Results and discussion

4.1 NVE Ensemble

For a preliminary study of the behavior of the system, it is easiest to use the NVE ensemble. Here we simply initialize the system with various temperatures and observe how it evolves by computing system energies, temperature, and diffusion coefficient. Without keeping the temperature constant, the system always equilibrates to about half of the initial set temperature soon after the start of the simulation. This is due to the exchange of kinetic and potential energy, while the total energy remains constant as can be clearly seen in Figure 5.

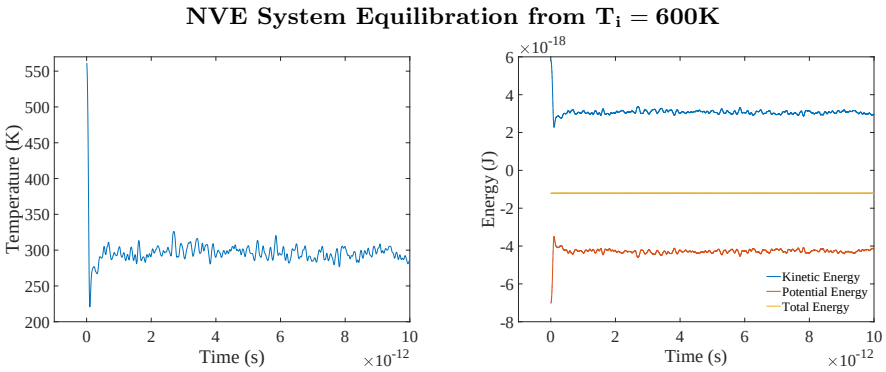

Figure 5: Equilibration of temperature and system energy of system initialized with $T = 600$K.

When the initial temperature is high enough to result in an equilibrium temperature larger than the melting point, the solid turns into a liquid. This can be observed by both the jumps in system energies and temperature and by viewing the system in *Ovito*. When a simulation was initialized at a temperature of 600K (corresponding to an equilibrium temperature of about 296K) or lower, the system stayed in the solid phase. The results for $T_i = 615$K are shown in Figures 6 and 7. The temperature of the system before melting appears to rise, so a fit was performed, resulting in an estimated melting temperature of 307K. The total system energy should stay constant, but it is observed here to rise slightly, perhaps due to the instability of the Verlet solver at the 20fs timestep. With the available computational resources, it was unpractical to use a smaller time step when simulating the system very close to the melting threshold, due to the number of steps necessary before the solid melts. It took the solid twice as long to melt with $T_i = 610$K (results not shown) than when the simulation was initialized with 615K and the estimated temperature before melting was 306K.

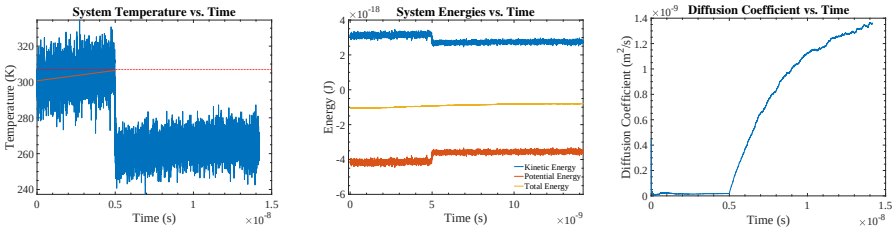

Figure 6: Temperature, energies, and diffusion coefficient vs. time for system initialized with $T_i = 610$K with the NVE ensemble.

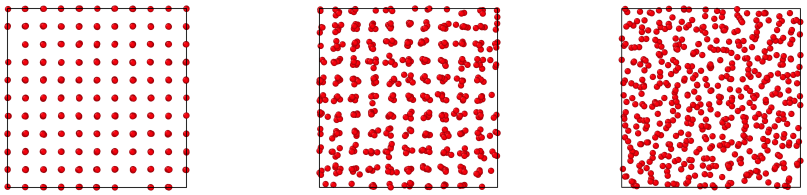

Figure 7: From left to right: system near start of simulation, distorted solid, liquid

Another way to identify the phase of a material, is to calculate its diffusion coefficient (Section 2.5). Since the atoms in a liquid are more free to travel, it will have a much higher diffusion coefficient than a solid. In order to search for the melting point, I plot the diffusion coefficient calculated at the end of each simulation run versus equilibrated temperature (Figure 8). From this plot, we can estimate that the melting point of argon is around 300K.

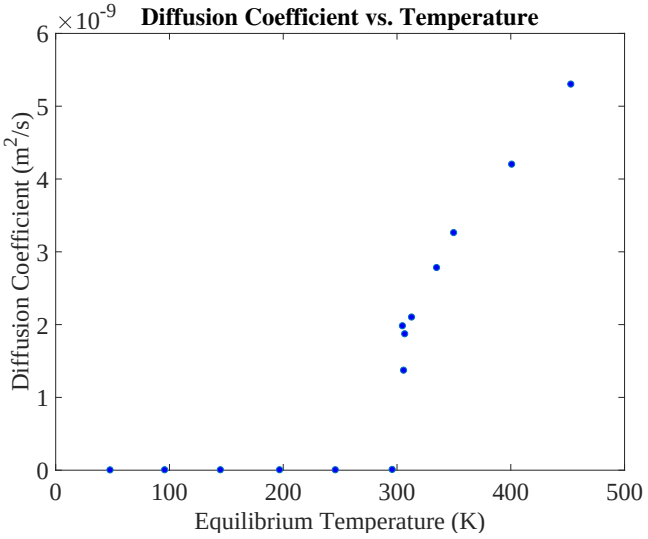

Figure 8: Diffusion coefficient as a function of equilibrated temperature for NVE ensemble simulations. For the cases where the solid melted, the equilibrated temperature is defined as average of the calculated temperatures right before melting.

4.2 Gradual heating

In order to determine the region at which melting occurs, I first ran a simulation with 108 atoms with initial temperature set to 160K (which quickly equilibrated to about 80K) and increased the temperature by δT=10^{-4}K at every time step

(δt=20fs). At every timestep the system's kinetic energy and from it, the instantaneous temperature, was calculated. The potential energy and diffusion coefficient was calculated periodically. The results are shown in the left column of Figure 9. The sharp jump in the potential and kinetic energies of the system is indicative of a first order phase transition. Looking in *Ovito*, I verify that at precisely this time the solid has turned into a liquid. Also the diffusion coefficient rises very rapidly at the transition point. At the transition, the kinetic energy and therefore temperature drops because changing the phase of a material requires an input of energy (latent heat). The potential energy sharply becomes less negative at the transition because the solid is in a deeper potential well than a liquid which has less strong interatomic interactions. A melting temperature of about 278K was found by fitting the temperature vs. time graph to find the temperature where the jump occurs. The fitting procedure is described in the Appendix.

Now, I ran the desired 500 atom simulation with a 20fs timestep, initializing the temperature closer to the estimated melting point. Initializing at 400K, gave an equilibrated temperature of about 197K and melting point of 305K. The results are shown in the central column of Figure 9. Next I decreased the timestep to δt=2fs and to reduce computational time, initialized at the higher temperature of 550K. Fitting the data as before, we get a melting temperature of 317K (right column of Figure 9). I also ran the same simulation with timestep of δt=20fs which gave a melting point of 310K. Therefore both timestep and temperature at which start gradual heating affect the calculated melting point.

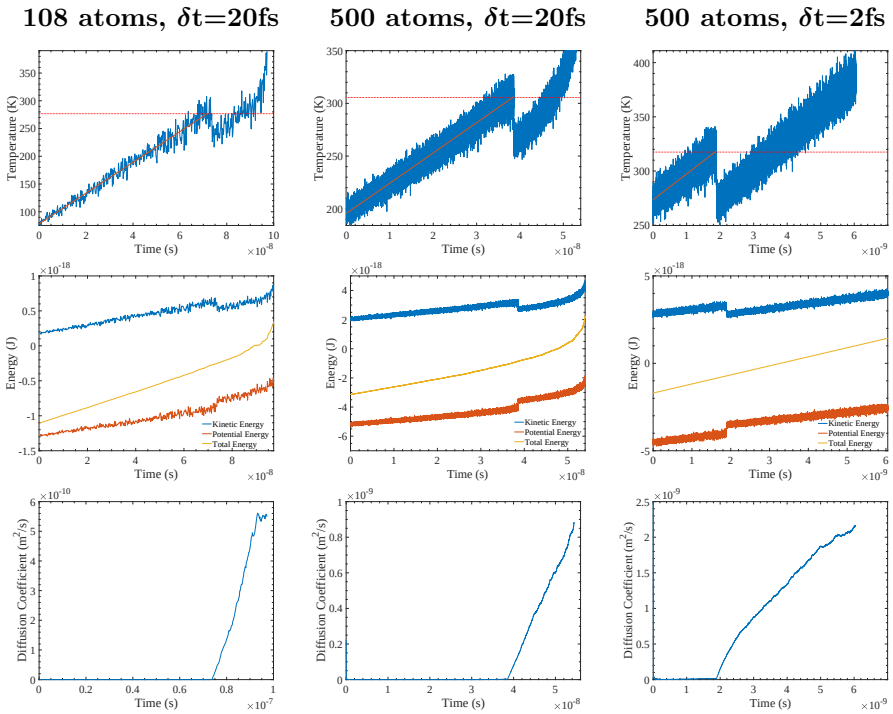

Figure 9: Results for gradual heating of 108 atoms with 20fs timestep and 500 atoms with 20fs and 2fs timesteps.

The $\delta t = 20fs$ graphs begin curving upward at higher temperatures because the simulation becomes unstable. If the simulation is continued, energies and temperature reaches extremely high values. This instability at higher tmeperature is due to that the system dynamics becomes significantly faster than the timestep being used. Therefore, an atom could have enough velocity to move into a position where it is nearly on top of another atom within 1 δt. Then at the next force evaluation, these atoms experience an extremely large force (recall the $1/r^{12}$ repulsive term), resulting them gaining extremely large velocities. This repeats and soon the simulation goes out of control. This is also visible in *Ovito* as a sudden explosion of the system!

Overall, I expect for the gradual heating simulation to overestimate the melting temperature. The reason is that the melting process takes time. We can see in *Ovito* that the atoms wiggle around with greater and greater intensity until the crystal structure gets distorted enough that it suddenly loses its order. So for example, 290K might be enough for the crystal to melt, but since the temperature is continously being increased, by the time the melting actually occurs, the system temperature may already be 305K. Since when using the 2fs timestep, the heating is ten times faster than with the 20fs step, it makes sense that the calculated melting temperature is higher.

4.3 NVT Ensemble

In order to more carefully check whether or not the solid melts at a certain temperature, I now keep the system's temperature constant by velocity rescaling (Section 3.5). Several temperatures near the expected melting point were tested. The results are shown below. Notice that it took more than ten times longer for the solid to melt at 290K than at 300K.

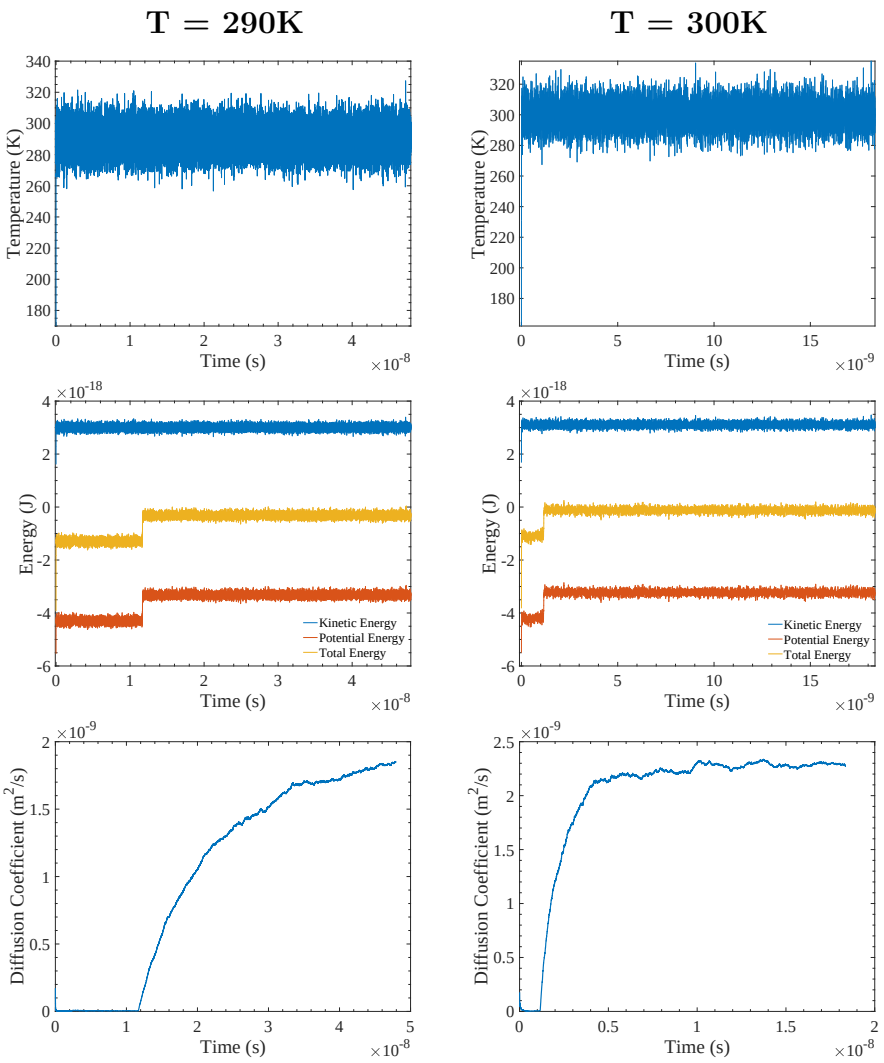

Figure 10: Results for NVT ensemble simulations at temperatures of 290K and 300K. Note that the time axis scales are different.

4.4 Comparison with literature

Argon has a melting point of 83.8K and boiling point of 87.3K [4]. One immediately wonders, why these simulation results are so drastically different. At first glance, it seems that I found a solid-liquid transition in a region where argon should be a gas! However, we must not forget that pressure affects the phase of a material and that the "standard" melting and boiling points are given at atmospheric pressure. Carefully considering how we constructed this simulation, I realize that the periodic boundary conditions artificially severely increase the pressure of the system. Most solids and liquids expand when heated but in our simulation we do not allow our system to expand, since it is kept at constant volume. With periodic boundary conditions, if an atom tries to leave the volume, it is artificially put back into the box on the other side. A solid which is not allowed to expand will experience an increasing pressure as the temperature rises. We can roughly estimate the pressure of a gas of 500 argon atoms confined to our simulation cell volume at 300K by using the ideal gas law $P = Nk_BT/V$. The pressure is about 1100 atmospheres or 11 kbar. If we look at the experimentally-determined phase diagram of argon (Figure 11) at these pressures, it is expected to be a solid or a liquid and the phase transition is expected to occur at 250-300K. This means that the melting temperature predicted by this simulation is reasonable. In order to study the melting of argon while controlling the system pressure, one can use the isobaric (NPT) ensemble where the system volume is rescaled to keep pressure constant.

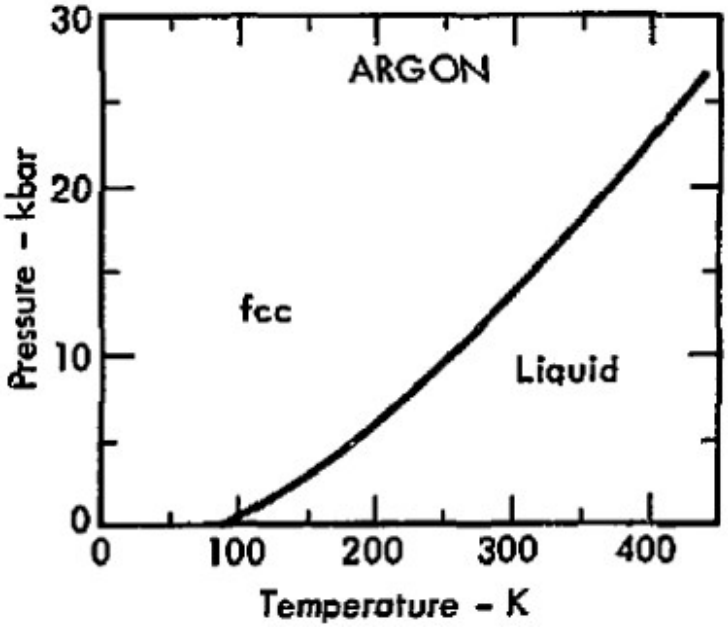

Figure 11: Pressure-temperature phase diagram of argon [11]

In order to further explore the effect of pressure on the simulation, I checked several limiting cases. One case is to initialize the system with triple the true lattice constant of argon while keeping the periodic boundary conditions. In this case, at a temperature of 200K, the system is now in the gas phase (Figure 12) instead of the solid phase we had when the lattice constant was smaller. This is because tripling the lattice constant increases the volume by a factor of 27 which decreases the pressure by the same factor. Therefore, now a rough estimate of the pressure from the ideal gas law would be 26 atm instead of the 730 atm we would have in the normal simulation at 200K.

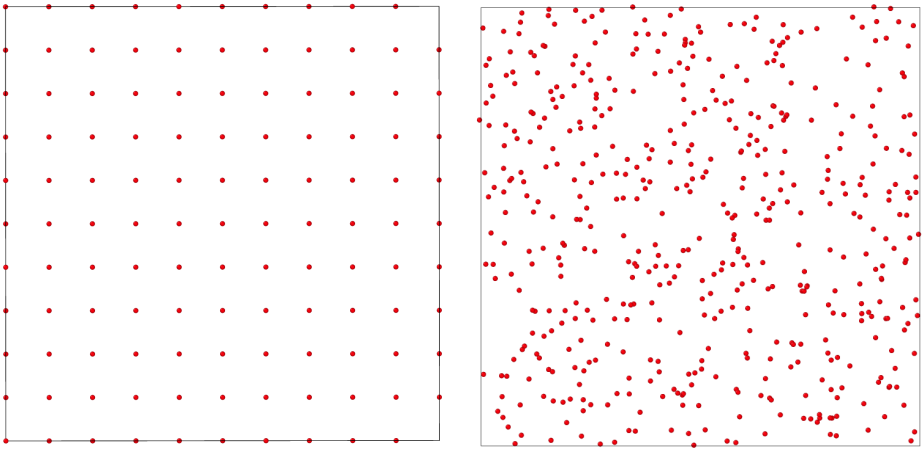

Figure 12: Initial state (left) and gaseous state (right) for NVT simulation at 200K initialized with tripling lattice constant.

Another case is the extreme of free boundary conditions where there are no limitations of where the atoms can travel. This is like having a small solid levitating in an ultra-high vacuum chamber. Such a system, very quickly melts and turns into a gas (Figure 13). After this, the simulation becomes unstable due to the particles having very large velocities which causes the velocity Verlet method to fail with the timestep I am using.

It is also useful to compare the calculated values of diffusion coefficient with literature. Both theoretical and experimental results show that the diffusion coefficient in liquid argon is of order of $10^{-9}m^2/s$ [12] which is consistent with the results found here.

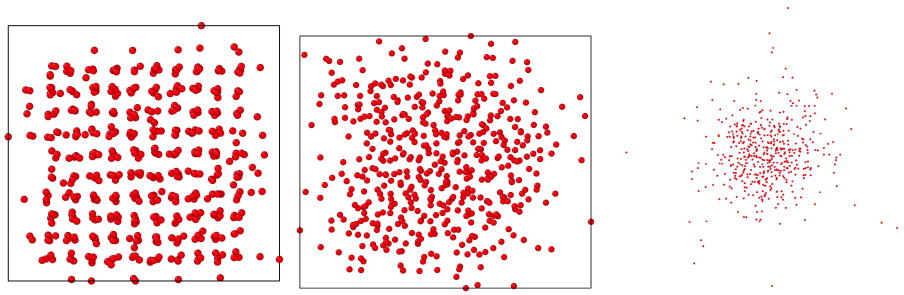

Figure 13: Melting/sublimating with free boundary conditions at 300K. The figure on the right is at a zoomed-out scale compared with the left two figures.

5 Conclusion

An object-oriented molecular dynamics code is developed to demonstrate the fundamental concepts of this computational approach and study the solid-liquid phase transition in argon. Three different approaches: the microcanonical ensemble (NVE), canonical ensemble (NVT), and gradual heating are tested. It was found that the NVT approach gave the lowest melting point. The overestimation of melting point in the NVE approach it is due to the system temperature dropping significantly as the solid is melting. The temperature can drop enough to be below the true melting point before the solid has a chance to melt. In the gradual heating approach, the overestimation is due to the heating is not stopped when the solid begins melting, so by the time the solid completes the transition, the temperature has already been increased. The upper bound for the melting point is found from the NVT ensemble to be 290K. This result is consistent with literature when system pressure is considered. With further simulations with larger system size and more MD steps, a lower melting point could potentially be found.

Supplementary Material

All programs and benchmarks calculations can be found in the GIT repository: `https://github.com/tgolubev/PHY905MSU/tree/master/Project4/molecular_dynamics`

References

[1] A. Rahman, Correlations in the Motion of Atoms in Liquid Argon. Phys. Rev. 136, A405, 1964.

[2] Ju Li, Basic Molecular Dynamics. Handbook of Materials Modeling, p. 565–588, 2005.

[3] Image from: https://d2gne97vdumgn3.cloudfront.net/api/file/dMxnY3mVQIK5z2rgVg23

[4] Element Collection, Inc http://periodictable.com

[5] Godehard Sutmann, Classical Molecular Dynamics, Published in: Quantum Simulations of Complex Many-Body Systems, Lecture Notes. John von Neumann Institute for Computing, Julich, NIC Series, Vol. 10, pp. 211-254, 2002

[6] Morten Hjorth-Jensen: Project4-2017 Computational Physics Spring 2017, https://github.com/CompPhysics/ComputationalPhysicsMSU/tree/master/doc/Projects/2017/Project4/MolecularDynamics

[7] Periodic Boundary Conditions https://en.wikipedia.org/wiki/Periodic_boundary_conditions

[8] Michael P. Allen, Introduction to Molecular Dynamics Simulation. Published in: Computational Soft Matter: From Synthetic Polymers to Proteins, Lecture Notes, John von Neumann Institute for Computing, Julich, NIC Series, Vol. 23, pp. 1-28, 2004.

[9] Morten Hjorth-Jensen. Ordinary differential equations https://compphysics.github.io/ComputationalPhysicsMSU/doc/pub/ode/pdf/ode-beamer.pdf, 2017

[10] A. Stukowski Visualization and analysis of atomistic simulation data with OVITO - the Open Visualization Tool Modelling Simul. Mater. Sci. Eng. 18 (2010), 015012 http://ovito.org/

[11] Young, David A. Phase diagrams of the elements. Lawrence Livermore National Lab, 1975 https://www.osti.gov/scitech/servlets/purl/4010212.

[12] Al-Matar et. al. Self diffusion coefficient of Lennard-Jones fluid using temperature dependent interaction parameters at different pressures. The Sixth Jordan International Chemical Engineering Conference, 2012. http://www.jeaconf.org/UploadedFiles/Document/bfdd24f0-be65-4f14-b7d5-f31e0ecceced.pdf

Appendix

Lennard Jones force calculation

The force calculation consists of two nested loops over the atom objects. The indices are setup to make use of Newton's third law and only calculate the force once per pair.

```
void LennardJones::calculateForces(System &system)
{
m_potentialEnergy = 0;  //reset potential energy

for(int current_index=0;current_index<system.num_atoms()-1;current_index++){
  Atom *current_atom = system.atoms(current_index);

  for(int other_index=current_index+1;other_index<system.num_atoms();
       other_index++){
     Atom *other_atom = system.atoms(other_index);

     //Apply minimum image convention
     vec3 displacement;
     for(int j=0;j<3;j++){
        displacement[j] = current_atom->position[j]-other_atom->position[j];
        if(displacement[j] >  system.systemSize(j) * 0.5){
           displacement[j] -= system.systemSize(j);
        }
        if(displacement[j] <= -system.systemSize(j) * 0.5){
           displacement[j] += system.systemSize(j);
        }
     }
     //precalculate
     double radiusSqrd = displacement.lengthSquared();
     double radius = sqrt(radiusSqrd);
     double total_force_over_r = m_twntyfour_epsilon*(2.0*pow(radius,-14.)-
          pow(radius,-8.));
     //find and set force components
     for(int j=0;j<3;j++) {
     current_atom->force[j] += total_force_over_r*displacement[j];
     other_atom->force[j] -= current_atom->force[j];
     }

     if(system.steps() % system.m_sample_freq ==0){
     //calculate potential energy every m_sample_freq steps
     m_potentialEnergy += m_four_epsilon*(pow(radius,-12.)-pow(radius,-6));
     }
}
}
```

Fitting procedure

Below is the Matlab code used to fit the temperature vs. time graphs for the gradual heating simulations to find the melting point. The basic idea is find the range of data on the left half of the graph which gives the best linear fit. This will almost always be the segment before the temperature jumps at the phase transition. The end of this data range will be near the melting temperature. Since the temperature oscillates, an average is taken over 10 point to the left.

```
minpts = 500;           %minimum # of points over which to do a fit
left_start_index = 2;   %skip the 1st data point since could be too low

for j = left_start_index+minpts:0.5*size(time)    %0.5*size(time) may need
     adjustment based on where jump is

time_data = time(left_start_index:j,1);
temp_data = temperature(left_start_index:j,1);
[~, temp_stat] = polyfit(time_data, temp_data, 1);    %find linear fit
temp_Rsquared(j,1) =  1 - temp_stat.normr^2 / norm(temp_data-mean(temp_data))
     ^2;    %calculate Rsquared and save values in column vector

end
```

```
[value,location] = max(temp_Rsquared);    %returns value and location (index)
      of maximum temp_Rsquared

%redo the fit which gave maximum R^2
time_data = time(1:location,1);
temp_data = temperature(1:location,1);    %make new x-variables column
[temp_fit, temp_stat] = polyfit(time_data, temp_data, 1);
temp_Rsquared_final =  1 - temp_stat.normr^2 / norm(temp_data-mean(temp_data)
    )^2
%take average of temp's surrounding phase transition area to get accurate
    estimate b/c temp. oscillates
phase_transition_temp = mean(temperature(location-10:location))
```

Additional NVE Results

Below is a plot of the ratio of equilibrated to initialization temperature. The ratio seems to vary with initial temperature, but it is unclear if there is any trend here.

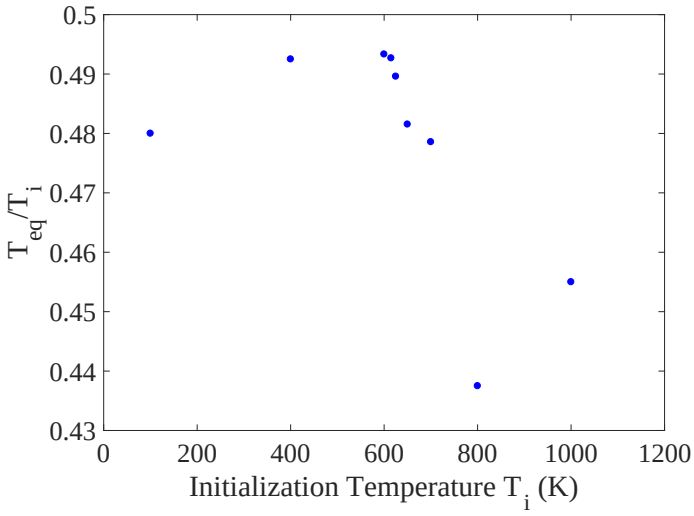

Figure 14: Equilibration temperature to initial temperature ratio as a function of initial temperature.