University of Oslo

FYS3150

Project 5

Escaping Lennard-Jones

A Molecular Dynamics adventure

Authors:
Erik Levén
Samuel Knudsen
Henrik Bjoner Lie

December 11, 2016

Contents

Abstract

The purpose of this project is to simulate the behaviour of argon in the phase transition from crystalline into liquid form. More precisely we try to find an approximation to the melting temperature of a system under high pressure and whose interactions can be adequately described by the Lennard-Jones potential. To analyze and visualize our results we will use the scientific visualization and analysis software Ovito[6].

With a density of 1823.25 $[kg/m^3]$, we find an upper limit to the temperature to be in the area around 290 K, but we conclude that it could possibly be found to be slightly lower using larger systems and longer simulation times than we were able to accomplish here. We thus recommend that more thorough research should be done on the subject.

I Introduction

Any element will undergo a phase transition when the temperature and/or pressure changes beyond/-below a critical point. The different phases solid, liquid and gas have different characteristics, not only on the macroscopic scale, but also on the atomic scale.

We want to explore the characteristics of these phases, by simulating a system of constant volume filled with a constant number of argon-atoms. The noble gas argon is used because its properties are such that we, to a good approximation, can emulate the atomic interactions of the atoms with the simple and popular Lennard-Jones potential, which we will describe more thoroughly later on. Molecular dynamics (MD) simulations like the one we are about to dive into are frequently used, not only to simulate small boxes of gas, but also to refine three-dimensional structures of proteins or study the movements of macromolecules in medicine, chemistry and biology. The method have made it possible to answer questions long unanswerable, and has given us great insight into the atomic-scale world.

This text will follow the development of a script setting up the MD simulation, using argon and the Lennard-Jones potential. We will take the reader through each step, carefully explaining the thoughts behind different choices made. Our ultimate goal is to describe and study the melting temperature of a system of argon atoms starting out in a crystalline solid state structure under high pressure.

Short overview of method

We will begin by representing our initial solid state as identical atoms in a three dimensional grid structure, where we will give our atoms random velocities from the Maxwell-Boltzmann distribution corresponding to an initial temperature T_0.

We will then use the Lennard-Jones potential to calculate the forces from the particles onto each other, and integrate these numerically through the Velocity Verlet method for a chosen amount of time steps.

By storing the positional values of the atoms for each time step in a file, we can finally visualize the process using a program called Ovito. In addition we will calculate and store the various thermodynamical properties such as temperature, kinetic and potential energy, and diffusion at each time step to a different file so that we can further analyze the systems development.

A flowchart of the programs process flow is provided to give a further overview of how our system is set up, and can be found under the "Flowchart"-section on page 14.

Units

As we will operate with tiny distances and masses, it is wise to define some more proper units than our standard SI-units for mass, distance, energy, temperature and time. Our units used in the program will be the following:

- 1 unit of mass = 1 u = $1.661 \cdot 10^{-27}$ kg
- 1 unit of length = 1.0 angstrom = $1.0 \cdot 10^{-10}$ m
- 1 unit of energy = $1.651 \cdot 10^{-27}$ J
- 1 unit of temperature = 119.735 K.
- 1 unit of time = $1.00224 \cdot 10^{-13}$ s

The units are deduced by setting the Boltzmann constant equal to one (in SI-units, $k_B = 1.3806 \cdot 10^{-23}[\frac{J}{K}]$) in the different equations governing the different parameters. Since temperature is often easier to read out in terms on Kelvin we have chosen to use Kelvin in the report but the above stated units in the code.

Ovito

Taken from the official Ovito website:

> "OVITO is a freely available visualization and data analysis software for atomistic data sets as output by large-scale molecular dynamics/statics and Monte-Carlo simulations. Its name is an acronym for Open Visualization Tool, emphasizing that flexibility and extensibility were important goals in the development of this software."[6]

Ovito lets us visualize our system, making it easier for us to understand and describe the physical processes evident in our data. The software works by taking an input file, processing the information through several analyzing and visualizing steps before finally rendering the result to screen.

II Initial state

Initial positions

We want our material to begin in a solid, crystalized state. There are many ways to represent this, and we have chosen one of the simplest ones - a so called face centered cubic (FCC) lattice, as this structure provides a local minimizer for Lennard-Jones type interaction for high densities[1].

The lattice consists of a number of unit cells - cubes with length b where each cube consists of four atoms. The local placement of the four atoms in a unit cell is as follows:

$$\begin{aligned} \mathbf{r}_1 &= 0\hat{\mathbf{i}} + 0\hat{\mathbf{j}} + 0\hat{\mathbf{j}} \\ \mathbf{r}_2 &= \frac{b}{2}\hat{\mathbf{i}} + \frac{b}{2}\hat{\mathbf{j}} + 0\hat{\mathbf{j}} \\ \mathbf{r}_3 &= 0\hat{\mathbf{i}} + \frac{b}{2}\hat{\mathbf{j}} + \frac{b}{2}\hat{\mathbf{j}} \\ \mathbf{r}_4 &= \frac{b}{2}\hat{\mathbf{i}} + 0\hat{\mathbf{j}} + \frac{b}{2}\hat{\mathbf{j}} \end{aligned} \tag{1}$$

where $\mathbf{r}_i$ is the local position of atom number i.

The complete system consists of $M \times M \times M$ such boxes, where M is an integer, where each atom now has 12 nearest neighbours (when applying periodic boundary conditions as explained below) each with a distance $\frac{b}{\sqrt{2}}$ away from the chosen atom. Figure 1 shows a graphical representation of the 12 nearest neighbour atoms.

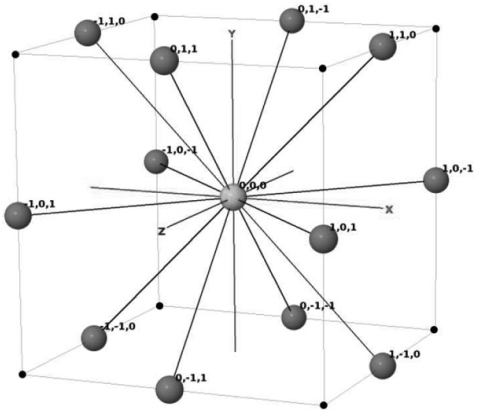

Figure 1: Figure showing the nearest 12 neighbors of an atom in the FCC lattice with b=2.

The constant b constitutes the length of each individual box. The entire length of the system in one dimension is therefore bM. In this project we have chosen a b-value of 5.26 [Å].

The initial state for 500 particles is visualized using Ovito in figure 2 and 3.

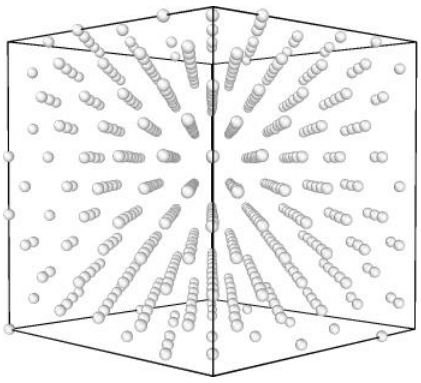

Figure 2: The initial crystalized state visualized in Ovito in three dimensions.

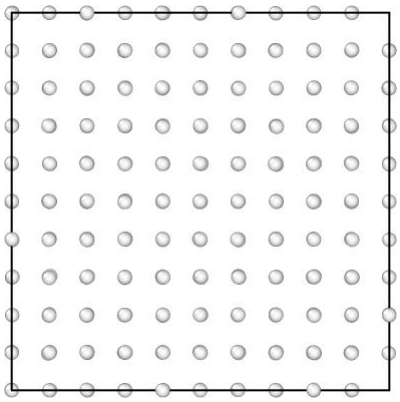

Figure 3: The inititial state viewed two dimensionally.

Initial velocities

In order to give our system a non-zero temperature, we need to provide our atoms with some momentum. We will distribute the velocities of the atoms randomly, using the famous Maxwell-Boltzmann distribution so that

$$P(v_i)dv_i = \Big(\frac{m}{2\pi k_B T_i}\Big)^{1/2} exp\Big(-\frac{mv_i^2}{2k_B T_0}\Big)dv_i, \tag{2}$$

where m is the mass of the atom, k_B is Boltzmann constant and T_0 is the temperature of choice.

T_0 is however not the actual temperature of the system as it is (most likely) not in equilibrium when we first begin calculations. This means that when the system does reach equilibrium, the temperature will most probably be different from T_0. T_0 is rather a measure of the initial internal kinetic energy we give the system.

Removing net momentum

The chance of ending up with a perfectly symmetric distribution of velocities in all dimensions is rather minuscule, which means that our system

will begin with a non-zero total momentum (as in, the center of mass will be moving). This could complicate the calculation of the key parameters later on, so we want our system to stay still.

We can remove this center of mass motion by calculating the net velocity vector, which is the average velocity of all the atoms, and then subtracting this vector from all the individual atoms' velocities.

Density

Using the lattice constant b in section "Initial positions" we can now calculate the density of the system. The density in units $[kg/m^3]$ can be calculated by the following equation:

$$d = \frac{4m}{b^3} \tag{3}$$

Here, m is the mass of an Argon atom in kg and b is the length in meters of one unit cell containing 4 atoms. This gives us a total density of 1823.25 $[kg/m^3]$.

Periodic boundary conditions

Because of limited computing power, the amount of atoms we can use in our simulation is very limited. This forces us to consider how to treat the atoms closest to the edges of the system, because the relative effect of these will be large for a small system. Remember that every atom should have 12 nearest neighbours, but the atoms on the edges will have fewer, and the unfortunate corner-atoms fewer still. By implementing what is known as periodic boundary conditions (PBCs) we can essentially simulate an infinitely large system through infinite repetition.

This works by having atoms that would otherwise leave our simulation-area instead slip through the "wall" and reappear on the opposite side. That is, if we have a two-dimensional system of size $L \times L$ with the origin at the lower left corner, put an atom at the origin and give it a velocity in the x-direction only, when this atom reaches x = L, we move it so that in the next time-step it is located at x=x-L, that is back to the origin in our case. This can easily be converted to three-dimensional systems.

III The L-J potential

The Lennard-Jones potential is commonly used to estimate the inter-atomic potential between two neutral molecules. The potential is defined as

$$U(r) = 4\epsilon\Big[\Big(\frac{\sigma}{r_{ij}}\Big)^{12} - \Big(\frac{\sigma}{r_{ij}}\Big)^{6}\Big] \tag{4}$$

where ϵ defines the depth of the potential well, σ is the finite distance at which the potential is zero and r is the distance between the two atoms. The $\frac{1}{r^{12}}$ part is the Pauli repulsion contribution in which the atoms move away from each other.

This force arises when the electrons of the two atoms are so close that their orbitals overlap. The $\frac{1}{r^6}$ part is the attractive van der Waal contribution, originating from dipole-dipole interactions. Those forces are fairly weak, but they dominate the bonding character of gases such as argon, which is why argon and Lennard-Jones are seen together frequently.[7]

Figure 4 shows a graphical representation of the Lennard Jones potential.

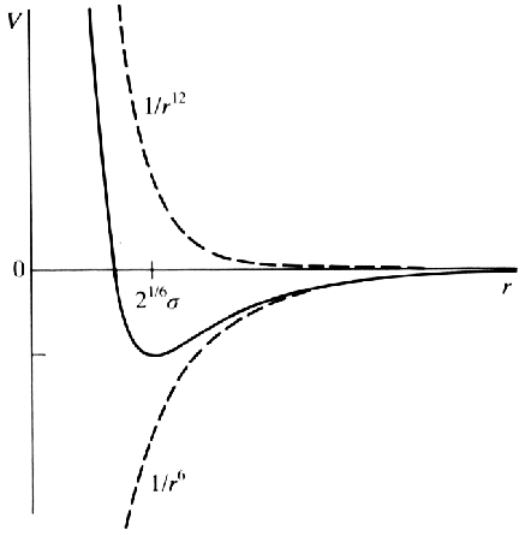

Figure 4: Figure showing the behaviour of the Lennard Jones Potential and how it's a sum of Pauli repulsion and the potential from the van der Waal force.

For this project we have chosen $\sigma = 3.405$ Å and $\frac{\epsilon}{k_b} = 119.8K$ which are considered the optimal values for argon simulations with a FCC lattice and a Lennard Jones potential. This makes sure we operate with high densities which suits the Lennard Jones model with a minimum image convention as described below, since we then do not have to take long-range forces into account. The force between

the particles is defined as

$$F(r) = -\frac{dU}{dr} \tag{5}$$

which when performing the derivation gives us

$$F(r) = \frac{24\epsilon}{r}[2(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}] \tag{6}$$

Minimum image convention

The minimum image convention is a way to ensure that the forces in our system is calculated correctly between an atom and the images of other atoms in the neighbouring "boxes". The problem this convention eliminates arises when two atoms is close to a wall, and one of them moves through said wall. The closest distance between them is then through the wall rather than across the system, where the atom in motion would now find itself thanks to PBCs.

Since we have already set up the periodic boundary conditions, we can visualize that each atom has six mirror atoms, one in each box surrounding the system we are actually in. The minimum image convention states that when calculating the force between an atom pair, starting with atom A, we choose the mirror of atom B that is closest to atom A.

What this essentially does is to say that the force between an atom pair is zero when the distance between the atoms is larger than $\frac{L}{2}$, where L is the box length. When decomposing in all three dimensions the code snippet looks like the following:

```
double dx = atom1.pos(0) - atom2.pos(0);
double dy = atom1.pos(1) - atom2.pos(1);
double dz = atom1.pos(2) - atom2.pos(2);

if ((dx) <= -L/2.0) dx += L;
if ((dx) > L/2.0) dx -= L;

if ((dy) <= -L/2.0) dy += L;
if ((dy) > L/2.0) dy -= L;

if ((dz) <= -L/2.0) dz += L;
if ((dz) > L/2.0) dz -= L;
```

Increasing efficiency

Since we are using numerical methods, computation time is an issue we have to address. As a way to make the code more efficient, we consider the 39 FLOPS we have the computer do when calculating $F(r)$. Performing this calculation for every atom pair results in a major CPU-time contributor. To speed up the calculation process without loosing accuracy we can use Newton's third law, which states that the force from particle A on particle B has the same magnitude but the opposite direction of the force from particle B on particle A.

This means that we can significantly reduce the amount of times we calculate $F(r)$ by storing the force from every calculation. The for-loop would look like

```
for (int i=0; i<numberOfAtoms; i++){
    Atom *atom1 = system.atoms()[i];
    for (int j=i+1; j<numberOfAtoms; j++){
        Atom *atom2 = system.atoms()[j];
```

which guarantees that we only iterate over each atom pair once. The force can now be stored with the following code snippet

```
    atom1->force[0] += force_scalar*dx;
    atom1->force[1] += force_scalar*dy;
    atom1->force[2] += force_scalar*dz;

    atom2->force[0] -= force_scalar*dx;
    atom2->force[1] -= force_scalar*dy;
    atom2->force[2] -= force_scalar*dz;
```

where force_scalar is our $F(r)$ and dx, dy and dz are the distances between the atoms in the x-, y- and z-direction respectively. This makes sure we only have to calculate the force between each atom pair once.

In addition to this we see that the force quickly decreases as the distance between the atoms increase. This indicates that we are performing many force calculations where the contribution is very close to zero. We thus want to set a boundary for our force, at which we will include a cut-off.

When studying figure 4 we see that in the area around $r = 2.5\sigma$ the derivative of the potential is very small. This is the r-value we will choose as our boundary, and it is also evidently a popular value for the cut-off [9]. The forces between atoms who have a distance larger than $r = 2.5\sigma$ between them will be considered as zero.

To avoid the problem of a non-continuous potential we will also have to "lift" the potential to make sure the value of $V(2.5\sigma) = 0$. This is done by subtracting the term $V(2.5\sigma)$ from the potential when we calculate the potential energy. The force is however not continuous due to the cut-off. This problem was deemed not essential in this particular project, all though it's might be worth investigating how this could effect the results.

IV Velocity Verlet

The Velocity Verlet algorithm is only one of many numerical methods used to integrate Newtons equation of motion. It was first used as early as the late 1700s, and has since been rediscovered many times, most recently by Loup Verlet, whose name is now associated with the algorithm, after he used it in molecular dynamics in the 1960s.

There are many similar algorithms often used on other problems than the ones covering molecular dynamics, for example the "Störmer-Verlet" algorithm, which were first used by Carl Störmer to calculate the trajectories of charged particles in a magnetic field. This algorithm does not make use of the velocity, but calculates the position at the next time-step through the previous two positions and the acceleration.

The Velocity Verlet algorithm is the more popular version. It is, as mentioned, based on Newtons second law, $\mathbf{F} = m\mathbf{a}$, and through this the acceleration, velocity and position of the particle in question. To derive the algorithm we make use of Taylor expansions, and look at how we can find an approximation to the position at a future time-step:

$$x(t+\Delta t) = x(t) + x'(t)\Delta t + \frac{x''(t)}{2}\Delta t^2 + \frac{x^{(3)}(t)}{6}\Delta t^3 + \frac{x^{(4)}(t)}{24}\Delta t^4 \quad (7)$$

we can also find the position at a previous time-step:

$$x(t-\Delta t) = x(t) - x'(t)\Delta t + \frac{x''(t)}{2}\Delta t^2 - \frac{x^{(3)}(t)}{6}\Delta t^3 + \frac{x^{(4)}(t)}{24}\Delta t^4 \quad (8)$$

adding equations 7 and 8 gives us the following expression for the second derivative of the position (the acceleration):

$$\mathbf{x}(t+\Delta t)+\mathbf{x}(t-\Delta t) = 2\mathbf{x}(t)+2\frac{\mathbf{x}''(t)}{2}\Delta t^2+\mathcal{O}(\Delta t^4)$$

$$\mathbf{x}''(t) = \frac{\mathbf{x}(t+\Delta t)+\mathbf{x}(t-\Delta t)-2\mathbf{x}(t)}{\Delta t^2} \quad (9)$$

where we use the Big O notation. $\mathcal{O}(\Delta t^4)$ thus means that the truncation error we make (so far) is of the magnitude of our time-step to the fourth power.

We recognize a problem here. To calculate the acceleration at the current time-step t we need to know both the position at the previous and the next time-step. Although these values can be given through initial conditions, the fact that we do need these conditions means that the algorithm is not "self-starting", and it can not be implemented on unknown systems as is. Notice also that equation 9 is not dependent of the velocity $\mathbf{v}(t) = \mathbf{x}'(t)$, which means we are unable to extract any information of the atoms velocities from this.

This equation is in itself the Verlet algorithm. To find the Velocity Verlet method, which we will use in our simulations, we need to include an expression for the velocity. This is done in much the same way as above, only we subtract equation 8 from 7, yielding:

$$\mathbf{x}'(t) = \frac{\mathbf{x}(t+\Delta t)-\mathbf{x}(t-\Delta t)}{2\Delta t} + \mathcal{O}(\Delta t^3) \quad (10)$$

We see that the velocity comes with a truncation error that goes like Δt^3. It can be shown that the global error in the Velocity Verlet algorithm is given by $\mathcal{O}(\Delta t^2)$. This comes from the fact that we are summing up the errors from every time-step. We have $\Delta t = \frac{1}{t_{total}} \Rightarrow t_{total} = \frac{1}{\Delta t}$, giving us the, somewhat rough, relation $t_{total}\mathcal{O}(\Delta t^3) = \frac{1}{\Delta t}\mathcal{O}(\Delta t^3) = \mathcal{O}(\Delta t^2)$. By implementing the velocity in our algorithm, and realising that we can find the acceleration through $\mathbf{F} = m\mathbf{a}$ and $\mathbf{F} = -\nabla U$ where U is a potential, in our case the Lennard-Jones potential, we can erase the first time-step problem, making the algorithm self-starting. We can now also extract information on the acceleration (the forces), the velocity and the position of the particle at the same time.

The general scheme of the Velocity Verlet algorithm is to first calculate the velocity at the next half time-step, $\mathbf{v}(t+\frac{1}{2}\Delta t)$, and then use this to calculate the position at the next whole time-step, $\mathbf{x}(t+\Delta t)$. The acceleration is then calculated at this new position using $\mathbf{a} = \frac{\mathbf{F}}{m}$, which means we are also updating the forces working in our system. Finally, we calculate the velocity at the next whole time-step through the acceleration and the velocity at a half time-step. The position and velocity at the new time step is now found, and another iteration can take place. In mathematical terms, it looks a little something like the following:

- $\mathbf{v}(t+\frac{1}{2}\Delta t) = \mathbf{v}(t) + \frac{1}{2}\mathbf{a}(t)\Delta t$
- $\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \mathbf{v}(t+\frac{1}{2}\Delta t)\Delta t$

- $\mathbf{a}(t+\Delta t) = \frac{\mathbf{F}(\mathbf{x}(t+\Delta t))}{m}$
- $\mathbf{v}(t+\Delta t) = \mathbf{v}(t+\frac{1}{2}\Delta t) + \frac{1}{2}\mathbf{a}(t+\Delta t)\Delta t$

Why do we use the Velocity Verlet instead of, say, any of the Euler algorithms? It can be shown that the error in the Euler methods are all of order one, while the error in Velocity Verlet is of order two, which makes Verlet a more accurate algorithm. It is, as described, also a self-starting algorithm which makes it more flexible. And while the general Euler-algorithm is easy to set up, the Velocity Verlet-method is very stable in comparison, and it preserves energy well (the energy increases over long time scales, but this increase is constant in time)[4]. Further, Velocity Verlet is not necessarily more time-consuming than your simple Euler-method.

The algorithm can be written in code as follows:

```
for(Atom *atom : system.atoms()) {
    atom->velocity += atom->force*0.5*dt
                      /atom->mass();
    atom->position += atom->velocity*dt;
}
system.applyPeriodicBoundaryConditions(N);
system.calculateForces();

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

where system.calculateForces updates the forces after every atom have been moved to their respective new positions, and system .applyPeriodicBoundaryConditions makes sure the PBC works as intended. The initial forces acting in our system are calculated before this snippet.

V Gathering data

We have a model, a numerical method and a system ready to go. We now only need to define the thermodynamical parameters we want to look at.

Kinetic energy

The kinetic energy is easy to calculate. We simply take the sum over the kinetic energy of all the atoms:

$$E_k = \sum_i^N \frac{1}{2} m_i \mathbf{v}_i^2 \tag{11}$$

where N is the total number of atoms in our system and m_i and $\mathbf{v}_i$ is of course the mass and velocity of atom number i.

Temperature

We can use the above to calculate the temperature of the system, by making use of the equipartition theorem, which states that every quadratic degree of freedom will contribute an energy $\frac{1}{2}k_BT$ at temperature T.[5] A quadratic degree of freedom includes any energy whose equation is a quadratic function of a coordinate or velocity component, for example the translational kinetic energy of an atom in the x-direction, $\frac{1}{2}m\mathbf{v}_x^2$. Other examples include vibrational and rotational energy. The equipartition theorem gives us the average kinetic energy (thermal energy) of a system in thermal equilibrium with N atoms through

$$\langle E_k \rangle = \frac{3}{2} NkT \tag{12}$$

where we have assumed that the rotational and vibrational energy of the atom can be neglected when compared to the translational motion which contributes $E = 3 \cdot \frac{1}{2}kT$ for every atom. This gives us for the temperature:

$$T = \frac{2}{3} \frac{\langle E_k \rangle}{Nk} \tag{13}$$

Diffusion

We also calculate the important diffusion-constant. Through the Einstein relation, which relates the mean square displacement (MSD) to the diffusion coefficient. We have

$$\langle r^2(t) \rangle = 6Dt \tag{14}$$

where

$$r^2(t) = \frac{1}{N} \sum_i^N r_i^2(t) \tag{15}$$

and $r_i^2(t) = |\mathbf{r}_i(t) - \mathbf{r}_i(0)|^2$ is, $\mathbf{r}_i(t)$ is the position of atom i at time t and further, D is the diffusion coefficient we are interested in given by

$$D = \lim_{t\to\infty} \frac{\langle r^2(t) \rangle}{6t} \tag{16}$$

It's important to use the infinity limit in the analytical expression due to the initial instability of the $\langle r^2(t) \rangle$ curve. As an infinite sum is impossible to calculate without an infinite amount of time, we work around this in our numerical calculations by disregarding the initial nonlinear phase of $\langle r^2(t) \rangle$ and thus drop the limit, yielding:

$$D = \frac{\langle r^2(t) \rangle}{6t} \tag{17}$$

When implementing the formula for $\langle r^2(t) \rangle$, we have to backtrack and change our code to make

sure we calculate the correct displacement. Since we are using periodic boundary conditions, $\mathbf{r}_i(t)$ could change drastically over small changes in time, when an atom moves through a wall. As an example, if an atom moves straight in the x-direction through a wall, the displacement vector $\mathbf{r}$ would suddenly completely change magnitude and direction. This will obviously cause problems when we calculate and analyze the diffusion constant.

To work around this problem, we include a new property to each atom, namely the initial position. We then make sure that the initial position change as an atom slips through any wall, in such a way that the displacement is correctly calculated. Going back to our example, this means that when the x-directed atom moves through the wall, not only is the position of this atom moved to the opposite side of our simulation area (teleported in the negative x-direction), but our initial position is moved into the neighbouring box located on the negative x-side of our simulation volume, so that the only change in $\mathbf{r}$ is the length of the vector per time, the correct magnitude of the displacement, and not its direction.

What diffusion tells us

Diffusion is in and of itself defined as a natural process in which molecules will intermingle due to their kinetic energy of random motion.[8] A popular example is to put food coloring in a glass of water. Initially the coloring will be confined in a small section of the larger system that is the water, but will, given enough time, distribute itself equally over the whole system.

From equation 17 we see that D have units area per time. If an atom moves in a straight line, never changing direction, D will grow linearly, and it can from this be understood as the area of a spherical surface which grows in time. From equation 15 we have that $r^2(t)$ is the average distance of our atoms from their initial starting position. This means, if we have a solid state at low enough temperatures, $r^2(t)$ would be small and rather constant, meaning the diffusion constant would approach zero as $t \Rightarrow \infty$, because the atoms are trapped in the lattice-configuration by the potential well they find themselves in.

In a liquid, however, the atoms are free to move around, not bound to any configuration. The distance of any one atom to its starting position would then grow in time, as the atom have the whole system (in our case an "infinitely" large one) to move into. Remember that we, in our simulation, give every atom a random initial velocity after the Maxwell-Boltzmann distribution, ensuring that (probably) no atom starts at rest.

The diffusion coefficient can then tell us what state our system is in. If we observe a D close to zero, we are in a solid state, but if $\langle r^2(t) \rangle$ grows linearly, D is constant and we have simulated a liquid state.

Conservation of energy

In order to make sure that our program behaves according to the laws of physics, we should make sure that the total energy is conserved. This means that if U is the sum of the potential energy of all the particles, and E_k is the kinetic energy as described earlier, then $U + E_k$ should be constant throughout the simulation. This is expected since the force we derive from the Lennard Jones potential is a conservative force where energy conservation is expected.

We find U of course by using our Lennard-Jones potential:

$$U = \sum_{i}^{N} \sum_{j=i+1}^{N} 4\epsilon \Big[\Big(\frac{\sigma}{r_{ij}} \Big)^{12} - \Big(\frac{\sigma}{r_{ij}} \Big)^{6} \Big] \qquad (18)$$

where r_{ij} is the distance between atom number i and j.

The actual harvest

In our program the class "statistics sampler" takes care of the gathering and saving of data during our simulation. For every iteration, it calculates the kinetic and potential energies, the temperature, the diffusion constant and the density, before saving these to file. The file is then analyzed and visualized through plots. Another file is also created: the .xyz-file. This only contains the positions of the atoms, and is used with Ovito to visualize the development of the system.

VI Results

An initial test

Before running the program, we need to initiate some values for our initial parameters. We'll let $m = 39.948$ u, which is the mass of an argon atom, the lattice constant $b = 5.26$ Å, $\epsilon/k_B = \epsilon = 1.0$ K and $\sigma = 3.405$ Å.

For the first simulation we'll give the system an initial thermal energy corresponding to $T_0 = 300$ K

(as described in the Initial Velocities section), and choose a time step $dt = 10^{-14}$ s (≈ 0.1 in our special units). The number of unit cells per dimension is 5, making a total of $N = 500$ atoms. We'll begin by running the simulation for 1000 time steps.

The first thing we check when running our program is that the conservation of energy criterion is satisfied. We'll also make sure that we're able to reach a state of equilibrium with a relatively stable temperature. We'll write out these values for each 100th time step as displayed by the output in table 1.

TS	T	E_k	U	$E_k + U$
1	312.03	1954.48	-3715.11	-1760.63
101	158.62	993.54	-2753.05	-1759.51
201	163.34	1023.1	-2782.30	-1759.2
301	171.32	1073.11	-2832.50	-1759.39
401	160.29	1004.03	-2762.97	-1758.94
501	168.85	1057.66	-2816.81	-1759.15
601	159.61	999.76	-2759.3	-1759.54
701	166.16	1040.77	-2799.96	-1759.19
801	162.05	1015.08	-2774.26	-1759.19
901	170.68	1069.11	-2828.47	-1759.36

Table 1: The temperature T and energy values of the system at time t of the simulation. The time and energy shown is in the units that our program uses, while the temperature is given in Kelvin. TS stands for time steps.

Our system seems to quickly reach a state of thermal equilibrium at around 165 K (with only 500 atoms we cannot really expect it to fluctuate much less). In addition, our total energy seems to be well conserved within numerical approximations.

We can take a look at the development in Ovito after our 1000 time steps, shown in figure 5 below.

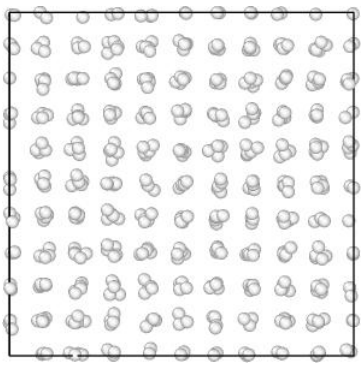

Figure 5: Snapshot of a micro state after about 1000 time steps with $T_0 = 300$ K.

We see that although the atoms are moving around a bit down in their own potential wells, the system seems to have retained its initial crystallized structure. It looks to be quite far from melting, so it seems unlikely that just giving it more time would change anything.

Let's try janking up the temperature a little bit. We'll set $T_0 = 900K$ and see what happens then. Figure 6 shows the result.

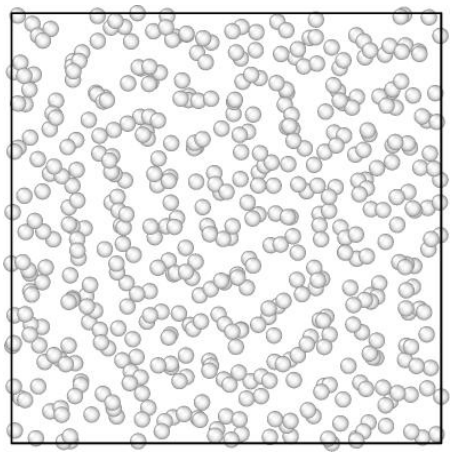

Figure 6: A microstate after about 1000 time steps with $T_0 = 900$ K. The lump of argon has indeed melted at this point.

The material is clearly melted at $T_0 = 900$ K, as the lattice-structure evident in figure 5 is now gone.

We can look at this data in more detail as we did for $T_0 = 300$ K in table 1, and we find that the equilibrium temperature is around 450 K, and also that here too the energy is conserved. The euilibrium temperature is about half that of T_0 as well.

We can take a further look at the relation T_{eq}/T_0 for a number of different $T_0's$ and see if this is a general trend. As we'll see below in figure 7, it indeed is.

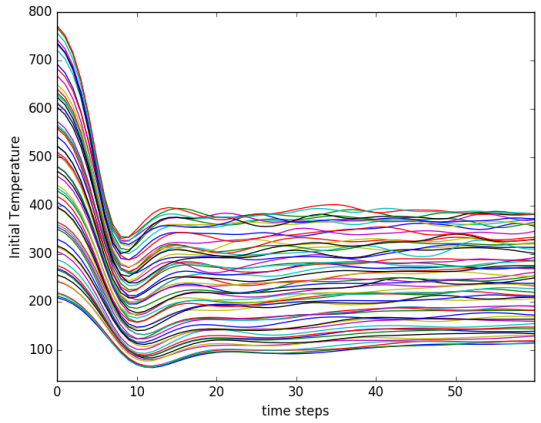

Figure 7: The temperature of the system as a function of time steps for a number of starting T_0 values. We see that all of them finds an equilibrium temperature at $T_0/2$.

Beginning the experiment

So in the previous section we found that the melting point should be found somewhere in the interval $300 \leq T_0 \leq 900$, corresponding roughly to an interval $160 \leq T_{eq} \leq 450$ for the equilibrium temperatures T_{eq}.

We can run a few quick simulations for initial thermal energies in this interval and plot the diffusion D as a function of temperature. Figure 8 shows the result when plotting as a function of T_0, while figure 9 is as a function of the equilibrium temperature. 10 000 time steps is used and the temperature interval is 10 K.

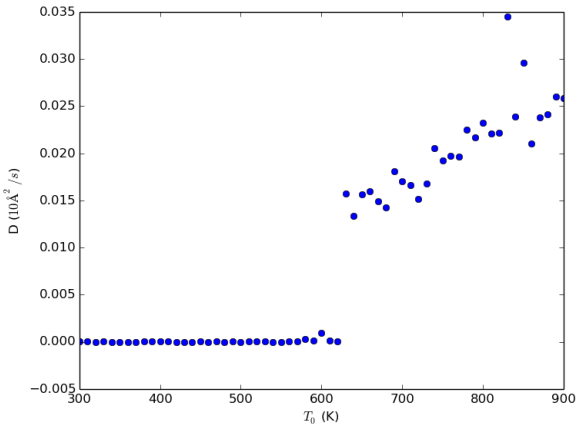

Figure 8: Diffusion as a function of our initial thermal energy represented by T_0. 500 atoms, 10 000 time steps.

This is interesting. We see in figure 8 that there is an immediate jump in diffusion around $T_0 = 630$. As explained earlier, a non-zero diffusion value indicates melting. This means that the internal energy corresponding to this value of T_0 causes a melting.

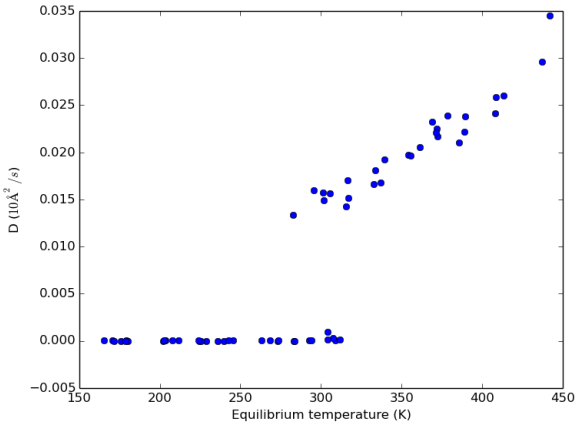

Figure 9: Diffusion as a function of equilibrium temperatures T_{eq}. 500 atoms, 10 000 time steps

To say something about the actual melting temperature of the system, we need to look at figure 9 above. The interesting part is that we see no zero diffusion values for temperatures above ca. 311 K, but at the same time we can see several non-zero diffusion for temperatures below this. This means that we have cases where one a solid and a melted state both have the same equilibrium temperature!

We believe that this can be explained by that the phase transition from solid to liquid in itself requires energy, and the system thus loses temperature when undergoing the transition. Our explanation for this follows in the next section.

A possible explanation

The particles are all in their own potential well (figure 4). In order for the system to melt, the particles need to climb the right side of this potential. As they do that though, they will lose kinetic energy. In the non-melting case, they will simply at some point fall down into the well again, and thus regain the kinetic energy.

In the melting case, however, they will escape to infinity and do not regain this kinetic energy (at least not many enough at any one time). This means that a macro state where the system just barely has enough energy for the particles to overcome the potential will necessarily have less internal kinetic energy after the phase transition, than

a system which has just below the required energy! Meaning, of course, that a system which barely melts will have a lower temperature after melting than a system which barely does not melt.

What all this implies is essentially that the melting point and freezing point are not the same. As we see from figure 9, any temperature above around 313 K causes melting, while any temperature below ca. 283 K will cause the system to retain the solid form.

Our melting point from figure 9 does therefore seem to be around 313 K. But we need to do more thorough research before we can conclude with anything.

A closer look

More time

At temperatures far away from the melting temperature, we can be rather quick about judging whether the state is melting or not. However, as T_{eq} approaches closer to the melting point, the system will need more and more time to go through phase transition process.

This is obvious. An ice cube will melt much more quickly in an oven heated up to $200°C$ than if you put it outside at $1°C$.

So to further study the melting point of our argon system, we should increase the number of time steps. We made it 50 000, and the result can be found in figure 10 below.

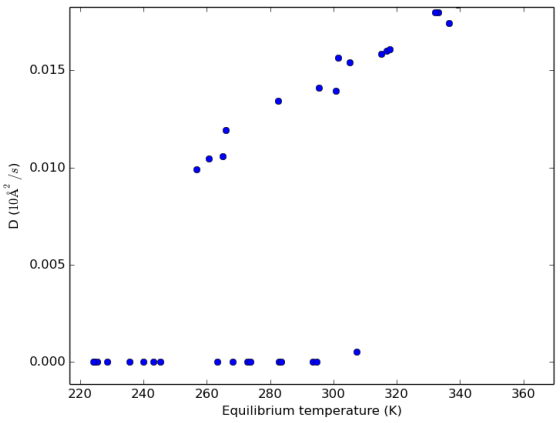

Figure 10: Diffusion coefficient value as a function of equilibrium temperature. 500 atoms, 50 000 time steps

We see now that the highest temperature for a diffusion close to zero is now at 307 K. The simulation time thus clearly affects the melting point.

More atoms

Another way to improve accuracy is to increase the size of the system. We turned the number of unit cells per dimension from 5 to 8, making a total of $N = 2048$ atoms. Due to the now much longer computing time, we turned down the number of time steps back to 10 000. The result is displayed in figure 11.

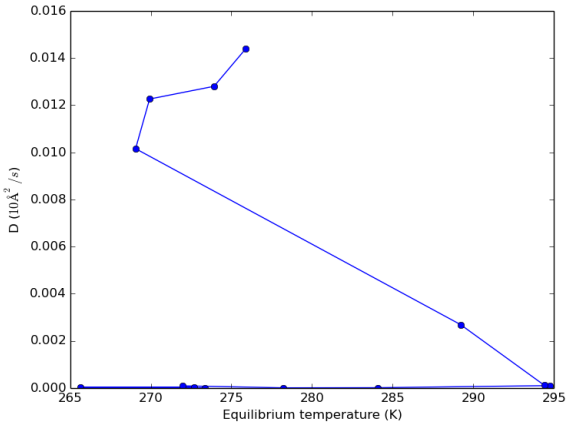

Figure 11: Diffusion coefficient value as a function of T_{eq}. 2048 atoms, 10 000 time steps.

We see now that our highest non-melting (diffusion = 0) temperature is at $T_{eq} = 295$ K. But we also see a point slightly higher up at around $T_{eq} = 289$ K. This could be a case where the system is *partly melted*, which could mean that the melting temperature is actually lower than 295 K, but that we need to increase the number of time steps for it to have time to melt.

In any case, it is obvious that both the size of the system and the simulation time increases the accuraccy of our statistics.

Looking even closer

It takes quite a lot of computer resources to run with very many atoms. With an amount of time steps significantly higher than 10 000, we need to limit the amount of temperatures we can look at.

We have run the simulation for a few selected initial internal energies. The following is the analysis of a simulation using 50 000 time steps, $N = 4000$ atoms and an initial thermal energy corresponding to $T_0 = 550$ K.

In figure 12 we see the state visualized after around 14 000 time steps:

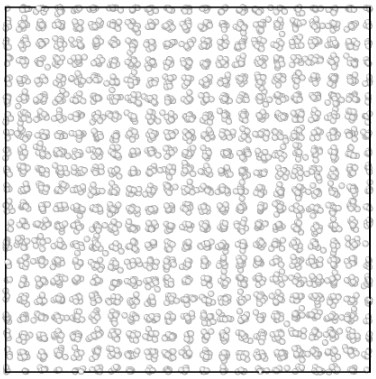

Figure 12: A microstate at around time step 14 000.

The system is still in a solid, crystalline state. Now, see out what happens after about 7000 time steps later (figure 13):

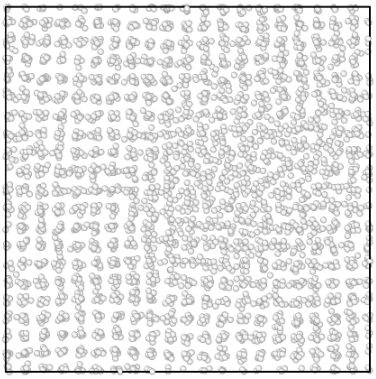

Figure 13: A microstate at around time step 20 000.

The system is clearly partly melting! This seems to fit very well with our hypothesis about partly melted states in the previous section. If we were to end the simulation now and look at the diffusion similarly as in figure 11, we would get a point that was in between between the two "point clusters". Around 10 000 timesteps later, the state looks like figure 14.

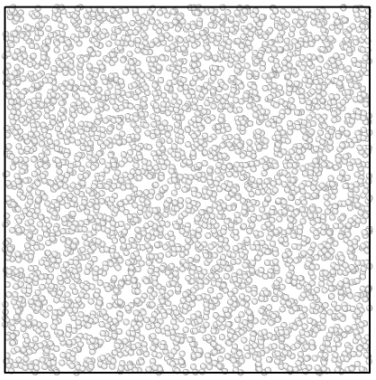

Figure 14: A microstate at around time step 30 000.

It took a total of 30 000 time steps for it to fully melt. Looking at the graph for the MSD $\langle r^2(t) \rangle$ in figure 15, we see the same thing.

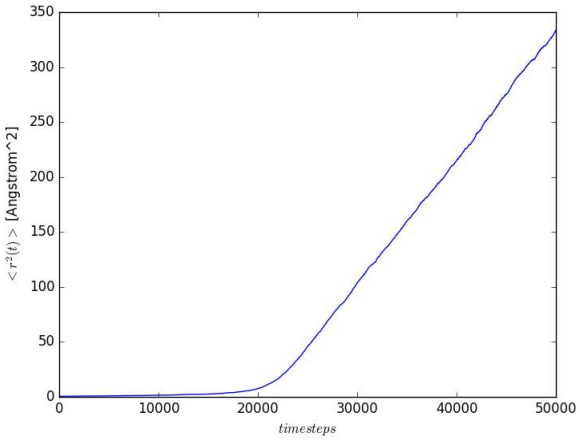

Figure 15: The mean square displacement value as a function of time steps. As explained, this is related to the diffusion of the system.

As we thought then, the results from figure 11 was not fully conclusive. This is a melted state which we wouldn't have seen from our first simulations with 10 000 time steps.

The equilibrium temperature for this simulation was found to be 244 K, but this was *after* melting so it doesn't tell us anything about the melting temperature (remember, the temperature decreases when melting). We calculated the temperature for the first 13 000 time steps to be 290 K, but since the system is not really in equlibrium then (as it is melting), meaning that the temperature is not actually properly defined, we're not sure if we can draw anything from this.

Highest non-melting temperature

We have run our simulation for a few more temperatures with even more time steps, but as snapshots from Ovito visualizations of these wouldn't really tell you anything new, we will omit putting them up here.

The highest equilibrium temperature that we found for a non-melting system of $N = 4000$ atoms, was at $T_{eq} = 289$ K. We initiated this with a $T_0 = 545$ K and ran it for 100 000 time steps. It is possible that this state would have undergone a phase transition even later on. It may also be that this temperature could cause a melting earlier with an even higher N, but due to our limited computer power, we are not able to verify this.

VII Conclusion

In this report we have described how we have developed a numerical model for simulating an isolated system of argon atoms under high pressure. We have found that the model indeed replicates a phase transition from solid to liquid form for sufficiently high temperatures.

The largest system that we were able to simulate for a large number of time steps was a system of $N = 4000$ atoms.

The highest equilibrium temperature we found with this that didn't melt, was at $T_{eq} = 289$ K.

The lowest T_0 value that did cause melting, was $T_0 = 550$ K, with an equilibrium temperature of 244 K. This *could* indicate a freezing point at around this temperature, but as we haven't actually simulated freezing, we cannot say that with any significant certainty.

We can with large certainty say that we have found an upper limit of the melting point of the system to be in the area around 290 K. But as the melting point in this model could seem to converge downwards with increasing number of particles, we can also be quite certain that a value slightly below this can be found using more computer power to simulate with more particles and for a longer time period.

We therefore recommend that more research should be done on this subject with larger systems and longer simulation times.

VIII Flowchart

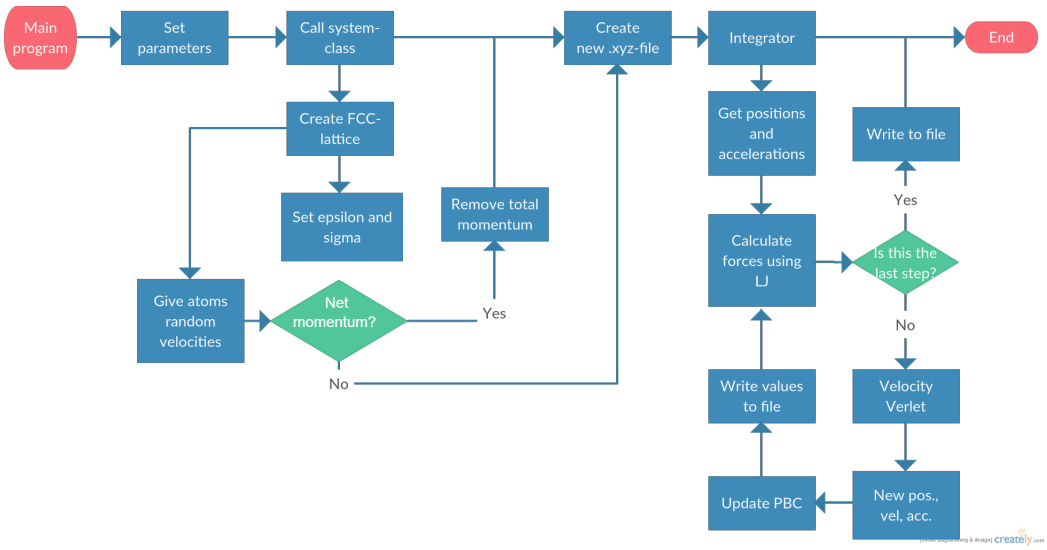

Figure 16: Flowchart showing the process flow of our program.

References

[1] Betermin, Laurent (2016). https://hal.archives-ouvertes.fr/hal-01401382

[2] Morten Hjorth-Jensen (2015), *Computational Physics, Lecture Notes Fall 2015*, University of Oslo

[3] Hannes Jónsson (2000), *Classical dynamics, lecture notes*, University of Washington, https://www.physics.drexel.edu/~valliere/PHYS305/Diff_Eq_Integrators/Verlet_Methods/Diffrntleqn3.pdf

[4] Ulf D. Schiller (2008), *An overview of integration schemes for molecular dynamics simulations*, http://www2.mpip-mainz.mpg.de/~andrienk/journal_club/integrators.pdf

[5] Daniel V. Schroeder, *An Introduction to Thermal Physics*, Essex, Pearson Education Limited, 2014.

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

[7] Furio Ercolessi, *The Lennard-Jones potential*, 1997, http://www.fisica.uniud.it/~ercolessi/md/md/node15.html, accessed 09.12.2016.

[8] R. Nave, *Diffusion and Osmosis*, http://hyperphysics.phy-astr.gsu.edu/hbase/Kinetic/diffus.html, accessed 09.12.2016.

[9] https://en.wikipedia.org/wiki/Lennard-Jones_potential

IX Source code

The source code in C++ and in Python can be found on the following GitHub address: https://github.com/erikalev/FYS3150/tree/master/Project%205