Using Molecular dynamics to calculate the melting temperature of a face centered cubic lattice crystal consisting of Argon atoms

Project 5, FYS-3150

Ina K. B. Kullmann, candidate nr: 20

Abstract

The aim of this project is to study Argon atoms in a face centered lattice crystal by developing a fully working Molecular dynamics code with the final goal to find the melting temperature of the solid.

We have started working from a skeleton code consisting of a network of classes and subclassen and implemented the remaining functions needed to calculate the statistical properties of the system. We have tested the implementation of the code by looking at the simulation in a visualization program and seen if the simulation behaved as expected according to our physical intuition. We have also compared two numerical integration methods and studyed the tecnical aspects and approximations of the numerical simulation.

Finally we found the melting temperature to be at $T = 275$ K with the lattice constant $b = 5.26$Å, a system with high particle density and pressure.

All source codes can be found at: `https://github.com/inakbk/molecular-dynamics-fys3150`.

Contents

1 Introduction

Molecular dynamics (MD) is a computer simulation method used to study atoms and molecule structure and movement. In a MD simulation the atoms or molecules are allowed to interact through a force given by a potential for a limited time. This makes it possible to study the system evolving in time. The applications of MD is many ranging from chemical physics, materials science and the modelling of biomolecules[1].

Molecular dynamics is often used to study stastistical properties of a system consisting of a large number of atoms or molecules and thus MD is a a type of N-body simulation. For systems that obey the ergodic hypothesis[2] the evolution of a single molecular dynamics simulation may be used to determine macroscopic thermodynamic properties of the system. This is because the time averages of an ergodic system correspond to microcanonical (NVE) ensemble averages[3]. In this paper we wil study one such NVE ensemple, a system with constant number of particles (N), constant volume (V) and constant total energy (E).

Often the main motivation to use Molecular dynamics is that it is not possible to determine the properties of the system analytically, mainly because of the large number of particles. Therefore one have to apply numerical methods to be able to solve the problem. One of the main limitation for the numerical simulation is the computer recources available limiting the number of particles and the system size. The latter problem can be solved by applying periodic boundary conditions so that it is possible to simulate a system with infinite size. The first problem might be solved with a very efficient code and good approximations. Another problem with the numerical simulation is cumulative errors in the numerical integration when evolving the system in time, often drift in the total energy. This error can be avoided by proper selection of parameters and integration algorithms. In this paper we will have a look at two numerical integration methods; the Euler-Cromer method and the Velocity Verlet integrator. We will also use a set of units that we will refer to as 'MD units' listed in appendix A to aviod calculations with small exponents.

In this project we will study the properties of a large system consisting of Argon atoms initially placed in a cubic lattice and try to determine the melting temperature of the solid. We are interested in the statistical properties of the system such as the mean square displacement and the diffusion constant. We will also have a quick look at the energy development in time. The individual motion of each particle is not of any physical interest, but we will look at the simulation with an visualization program to gain physical understanding of the system and to verify the implementation of the models and algorithms.

This report is organized into three parts. First we will look at the theory and the equations giving the physical properties of the system. Then we will look at the numerical methods and approximations used, the structureof the source code and program flow. But most importantly we will study the results obtained.

[1] `https://en.wikipedia.org/wiki/Molecular_dynamics#Areas_of_application_and_limitations`, 11.dec 12:00

[2] `https://en.wikipedia.org/wiki/Ergodic_hypothesis` 11.dec 11:30

[3] `https://en.wikipedia.org/wiki/Molecular_dynamics` 3.dec 11:25

2 Theory

2.1 Creating the lattice

We want to start out the simulation with the atoms in a crystal structure and will choose the face-centered cubic (FCC) lattice. Later we will learn that the FFC lattice is a stable structure for the Lennard-Jones potential that we will use in the simulation. The lattice is build up by unit cells, a group of attoms, so that a larger system can be created by repeating these cells in space. The FFC lattice unit cell is of size bÅ and consists of four atoms with local coordinates

$$\mathbf{r}_1 = 0\hat{\mathbf{i}} + 0\hat{\mathbf{j}} + 0\hat{\mathbf{k}}, \tag{1}$$

$$\mathbf{r}_2 = \frac{b}{2}\hat{\mathbf{i}} + \frac{b}{2}\hat{\mathbf{j}} + 0\hat{\mathbf{k}}, \tag{2}$$

$$\mathbf{r}_3 = 0\hat{\mathbf{i}} + \frac{b}{2}\hat{\mathbf{j}} + \frac{b}{2}\hat{\mathbf{k}}, \tag{3}$$

$$\mathbf{r}_4 = \frac{b}{2}\hat{\mathbf{i}} + 0\hat{\mathbf{j}} + \frac{b}{2}\hat{\mathbf{k}}. \tag{4}$$

where $b = 5.26$Å is the lattice constant. We want to construct a larger system with $N \times N \times N$ such unit cells next to each other. The orgin of unit cell (i, j, k) is

$$\mathbf{R}_{i,j,k} = i\hat{\mathbf{u}}_1 + j\hat{\mathbf{u}}_2 + k\hat{\mathbf{u}}_3, \tag{5}$$

where $i = 0, 1, ..., N_x - 1, j = 0, 1, ..., N_y - 1, k = 0, 1, ..., N_z - 1$. The unit vectors of the unit cells are scaled with the lattice constant b so that

$$\hat{\mathbf{u}}_1 = b\hat{\mathbf{i}}, \quad \hat{\mathbf{u}}_2 = b\hat{\mathbf{j}}, \quad \hat{\mathbf{u}}_3 = b\hat{\mathbf{k}}. \tag{6}$$

The coordinates of particle one to four is then given by:

$$\begin{aligned} x_1 &= 0 + ib \\ y_1 &= 0 + jb \\ z_1 &= 0 + kb \end{aligned} \qquad \begin{aligned} x_2 &= \frac{1}{2} + ib \\ y_2 &= \frac{1}{2} + jb \\ z_2 &= 0 + kb \end{aligned} \qquad \begin{aligned} x_3 &= 0 + ib \\ y_3 &= \frac{1}{2} + jb \\ z_3 &= \frac{1}{2} + kb \end{aligned} \qquad \begin{aligned} x_4 &= \frac{1}{2} + ib \\ y_4 &= 0 + jb \\ z_4 &= \frac{1}{2} + kb \end{aligned}$$

In figure 1 we see an illustration of two unit cells placed side by side. A cubic lattice is formed by placing $N \times N \times N$ such unit cells side by side.

Each of the sides of the simulation box has the length $L = bN$ so that the size of the whole system or the volume of the cubic lattice is given by $V = L^3 = (bN)^3$ which is constant as long as b and N is constant. The number of unit cells in each direction N is also a measure of the number of particles in the system because it is 4 particles in each unit cells. The total number of particles in the simulation box is therefore $4N^3$. The mass density of the system is then:

$$\rho_m = \frac{M_{tot}}{V} = \frac{4 * N^3 * m}{(bN)^3} = \frac{4m}{b^3}$$

where m is the mass of one atom and is equal for all the atoms. The particle density is then:

$$\rho = \frac{4N^3}{(bN)^3} = \frac{4}{b^3}$$

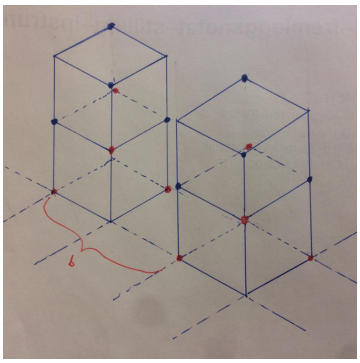


Figure 1: An illustration of the desired crystal structure when simulation an Argon crystal. The length of the sides of the unit cells used in this paper have half the length of the yellow box marked in the drawing.

We can see that both the mass and particle density will be constant as long as the mass and the lattice constant is held constant in the simulation. We therefore see that we have constructed a NVE ensemble since the number of particles and the volume is constant in the simulation. The system will not interact with any surroundings implying a conserved total energy.

2.2 Initial velocities

The atoms are usually given velocities according to the Maxwell-Boltzmann distribution:

$$P(v_i)\mathrm{d}v_i = \left(\frac{m}{2\pi k_B T}\right)^{1/2} \exp\left(-\frac{mv_i^2}{2k_B T}\right) \mathrm{d}v_i, \tag{7}$$

where m is the mass of the atom, k_B is Boltzmann's constant and T is the temperature. We recognize this as a normal distribution with zero mean and standard deviation $\sigma = \sqrt{k_B T/m}$.

The Maxwell-Boltzmann distribution describes particle speeds of ideal gasses where the particles move freely inside a stationary container. An ideal gass is an simplification of real gasses where the particles is assumed to not interact with one another except for very brief collisions in which they exchange energy and momentum with each other or with their thermal environment. The system of particles is also assumed to have reached thermodynamic equilibrium[4].

In real gases and in the Argon gass that we will study there are various effects such as van der Waals interactions. These effects imply that the gass does not fulfill the asumptions of an ideal gas. However the ideal gas is often a good approximation implying that the Maxwell speed distribution is also an good approximation. The distribution also depends on the temperature of the system and the mass of the particle which is beneficial in our case. We will therefore use the Maxwell-Boltzmann distribution to give initial velocities to the particles in the simulation.

Using the Maxwell-Boltzmann distribution to give initial velocities to the particles will result in a nonzero net momentum in the system. We will therefore remove this total momentum right after the initial velocities have been given.

[4] `https://en.wikipedia.org/wiki/MaxwellâĂŞBoltzmann_distribution` 11.dec 12:30

The momentum in a given direction i for one atom is $p_i = mv_i$ where the mass is m and v_i is the velocity in direction i. The total momentum in one direction, assuming that all particles have the same mass, is given by:

$$p_{tot} = m(v_{i,1} + v_{i,2} + v_{i,3} + \cdots + v_{i,n})$$

where n is the total number of atoms in the simulation.

We want to remove this momentum evenly from all the particles. The momentum we want to remove from each particle is then:

$$p_{rm} = \frac{p_{tot}}{n}$$

so that p_i' is the momentum for one particle after removing the total momentum:

$$p_i' = p_i - p_{rm}.$$

If we write this out

$$\begin{aligned} mv_i' &= mv_i - \frac{m(v_{i,1} + v_{i,2} + v_{i,3} + \cdots + v_{i,n})}{n} \\ \Rightarrow v_i' &= v_i - \frac{v_{i,1} + v_{i,2} + v_{i,3} + \cdots + v_{i,n}}{n} \\ &= v_i - \overline{v_i} \end{aligned}$$

we see that to remove the total momentum one only have to substract the average velocity from each of the particles velocity.

2.3 Periodic boundaryconditions (PBCs)

To avoid problems with boundary effects and to simulate a system with infinite size we will apply periodic boundary conditions. This has a great analogy to 'old' video games such as Snake 2. If the snake head passes through one side of box, it re-appears on the opposite side with the same velocity. In the system of atoms this would mean that if an atom should leave the simulation box at one side it should enter on the oposite side with the same values for the physical parameters as it had before it left. This also implies that an attom at the edge of the box will interact with an atom at the opposite side of the box so that every atom have the same number of 'neighbours'.

When using PBCs the size of the simulation box will be constant because particles that move outside the box will be placed inside again on the oposite side of the box. But the size of the simulation box must also be large enough to prevent unphysical behaviour. If the box is too small one unit cell might interact with itself. In the Snake analogy this would mean that the "head" interacts with or bites its own "tail" trough the wall which is allowed in the game, but not very physical. Thus the box size have to be large enough relative to the size of a unit cell, length of the simulation and the desired accuracy[5].

[5] `https://en.wikipedia.org/wiki/Periodic_boundary_conditions#Practical_implementation:_continuity_and_the_minimum_image_convention` 11.dec 13:30

2.4 Interactions and forces: The Lennard-Jones potential

In this project we will use the Lennard-Jones potential which is a mathematically simple model that approximates the interaction between a pair of neutral atoms or molecules. Due to its computational simplicity, the Lennard-Jones potential is used extensively in computer simulations even though more accurate potentials exist[6].

The potential calculates the energy between two atoms i and j as

$$U(r_{ij}) = 4\epsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right], \tag{8}$$

where $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$ is the distance from atom i to atom j, ϵ is the depth of the potential well (units energy) and σ is the distance at which the potential is zero. The parameters in the potential can be fitted to reproduce experimental data. For argon, optimal values of the parameters are:

$$\frac{\epsilon}{k_B} = 119.8 \text{ K}, \sigma = 3.405 \text{ Å}. \tag{9}$$

With these parameters the L-J potential reproduces equilibrium thermodynamic properties that are in good agreement with experimental values for argon[7].

The force is calculated by taking the negative gradient of the potential

$$\mathbf{F}(r_{ij}) = -\nabla U(r_{ij}), \tag{10}$$

with x-component (the other components are calculated the same way)

$$F_x(r_{ij}) = -\frac{\partial U}{\partial r_{ij}}\frac{\partial r_{ij}}{\partial x_{ij}}, \tag{11}$$

where x_{ij} is the x-component of $\mathbf{r}_{ij}$. Calculating the differensials separately:

$$\begin{aligned}
\frac{\partial U}{\partial r_{ij}} &= \frac{\partial}{\partial r_{ij}} 4\epsilon\Big[\Big(\frac{\sigma}{r_{ij}}\Big)^{12} - \Big(\frac{\sigma}{r_{ij}}\Big)^{6}\Big] = 4\epsilon\Big[\sigma^{12}\frac{\partial}{\partial r_{ij}} r_{ij}^{-12} - \sigma^6 \frac{\partial}{\partial r_{ij}} r_{ij}^{-6}\Big] \\
&= 4\epsilon\Big[-12\sigma^{12} r_{ij}^{-13} - (-6)\sigma^6 r_{ij}^{-7}\Big] = \frac{24\epsilon}{\sigma}\Big[\Big(\frac{\sigma}{r_{ij}}\Big)^{7} - 2\Big(\frac{\sigma}{r_{ij}}\Big)^{13}\Big] \\
r_{ij} &= |\mathbf{r_i} - \mathbf{r_j}| = |x_i\hat{\mathbf{i}} + y_i\hat{\mathbf{j}} + z_i\hat{\mathbf{k}} - x_j\hat{\mathbf{i}} + y_j\hat{\mathbf{j}} + z_j\hat{\mathbf{k}}| = |(x_i - x_j)\hat{\mathbf{i}} + (y_i - y_j)\hat{\mathbf{j}} + (z_i - z_j)\hat{\mathbf{k}}| \\
&= \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} = \sqrt{x_{ij}^2 + y_{ij}^2 + z_{ij}^2} \\
\frac{\partial r_{ij}}{\partial x_{ij}} &= \frac{\partial}{\partial x_{ij}}\sqrt{x_{ij}^2 + y_{ij}^2 + z_{ij}^2} = \frac{1}{2r_{ij}} \cdot 2x_{ij} = \frac{x_{ij}}{r_{ij}}
\end{aligned}$$

gives

$$\begin{aligned}
F_x(r_{ij}) &= -\frac{24\epsilon}{\sigma}\Big[\Big(\frac{\sigma}{r_{ij}}\Big)^{7} - 2\Big(\frac{\sigma}{r_{ij}}\Big)^{13}\Big] \cdot \frac{x_{ij}}{r_{ij}} \\
&= \frac{24\epsilon}{\sigma^2}\Big[2\Big(\frac{\sigma}{r_{ij}}\Big)^{14} - \Big(\frac{\sigma}{r_{ij}}\Big)^{8}\Big] x_{ij}
\end{aligned}$$

[6] `http://en.wikipedia.org/wiki/Lennard-Jones_potential` 11.dec 13:40

[7] `https://github.com/CompPhysics/ComputationalPhysics1/blob/master/doc/Projects/Project5/project5_md.pdf` 11.dec 13:45, page 4

The force in y and z direction can be calculated the same way giving a total force

$$\begin{aligned}\mathbf{F} &= F_x\hat{\mathbf{i}} + F_y\hat{\mathbf{j}} + F_z\hat{\mathbf{k}} \\ &= \frac{24\epsilon}{\sigma^2}\Big[2\Big(\frac{\sigma}{r_{ij}}\Big)^{14} - \Big(\frac{\sigma}{r_{ij}}\Big)^{8}\Big]\Big(x_{ij}\hat{\mathbf{i}} + y_{ij}\hat{\mathbf{j}} + z_{ij}\hat{\mathbf{k}}\Big).\end{aligned}$$

2.5 Physical and statistical properties of the system

The potential energy for one particle-pair $U(r_{ij})$ is given by the Lennard-Jones potential introduced in section 2.4. The total potential energy V in the system is computed by summing over all pairs of atoms (counting each pair only once)

$$V = \sum_{i>j} U(r_{ij}). \tag{12}$$

The kinetic energy is defined as

$$E_k = \sum_{i=1}^{N_{\text{atoms}}} \frac{1}{2} m_i v_i^2, \tag{13}$$

where m_i and v_i is the mass and the speed of atom i. The total energy of the system is then $E = V + E_k$. We will also calculate an estimate of the temperature through the equipartition theorem[8]

$$\langle E_k \rangle = \frac{3}{2} N_{\text{atoms}} k_B T. \tag{14}$$

We can use this to define an *instantaneous* temperature

$$T = \frac{2}{3} \frac{E_k}{N_{\text{atoms}} k_B}. \tag{15}$$

We use the Einstein relation that relates the so called mean square displacement (MSD) $\langle r^2(t) \rangle$ to the diffusion constant D

$$\langle r^2(t) \rangle = 6Dt, \tag{16}$$

where t is time. The square displacement for atom i is calculated as

$$\begin{aligned} r_i^2(t) &= |\mathbf{r}_i(t) - \mathbf{r}_i(0)|^2 && (17)\\ &= (x_i(t) - x_i(0))^2 + (y_i(t) - y_i(0))^2 + (z_i(t) - z_i(0))^2 && (18)\\ &= x_i^2 + y_i^2 + z_i^2, && (19)\end{aligned}$$

where $\mathbf{r}_i(t)$ is the position of atom i at time t. The diffusion constant is then given by

$$D = \frac{\langle r^2(t) \rangle}{6t} = \frac{r_1(t)^2 + r_r(t)^2 + \cdots + r_n(t)^2}{6tn}$$

for a given initial temperature T_i.

The diffusion constant of the system can be used to measure the melting temperature. Atoms in a solid will not move much giving a mean square displacement close to zero and thus a diffusion constant close to zero. When the solid melts we would expect the MSD and the diffusion constant to increase from zero revealing the melting temperature of the crystal.

[8] See for example D. Schroeder's *An Introduction to Thermal Physics* for details.

3 Numerical methods and approximations

3.1 Folder structure

The source code in this project was developes from a skeleton code that can be found at `https://github.com/andeplane/molecular-dynamics-fys3150/tree/master`. This skeleton code includes a structure of classes and subclasses that creates particles uniformely distrubuted inside the box with initial velocities given by the Maxwell-Boltzmann distrubution. The code does not account for interactions between the particles and there is no periodic boundary conditions, but the structure needed to implement these methods are included. The skeleton code also integrates with the Euler-Cromer method and converts between the SI and MD units.

The source code developed in this project can be found at: `https://github.com/inakbk/molecular-dynamics-fys3150`. The source code is structured in the following way (boldface for folder):

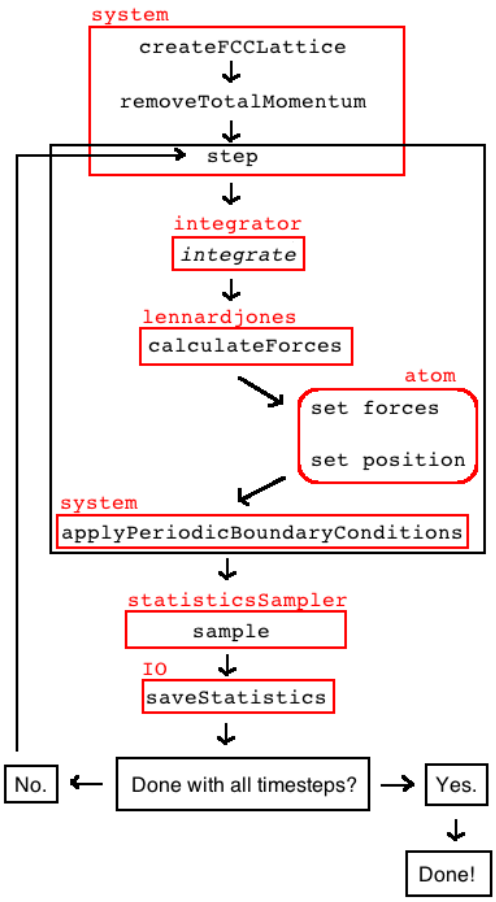
- **integrators**
  - eulercromer.cpp (subclass)
  - integrator.cpp
  - velocityverlet.cpp (subclass)
- **math**
  - random.cpp
  - vec3.cpp
- **potentials**
  - lennardjones.cpp (subclass)
  - potential.cpp
- atom.cpp
- io.cpp
- main.cpp
- statisticssampler.cpp
- system.cpp
- unitconverter.cpp

there is also corresponding folders and files with header files and python scripts for plotting the data `run_plot'...'.py`.

3.2 Program flow

The execution flow from `main.cpp` can briefly be described as follows:

- Recive standard input parameters or input from command line
- Create FFC lattice
- Set potential and integrator
- Remove total momentum
- Open files to save data; `movie.xyz`, `statistics_file.txt` (or unicque filenames with inputparameters).
- For all timesteps:
  - Print positions to `movie.xyz`
  - Do a timestep (integrate)
  - Sample data from system
  - Print data to `statistics_file.txt`
  - Print data in terminal every 100 timesteps
- Close `movie.xyz`
- Save execution time to `statistics_file.txt`
- Close `statistics_file.txt`



The figure roughly illustrates the program flow, what the different functions (black) do and which classes (red) they belong to. We can see that the first step is to create the FFC lattice so the positions and initial velocities are set for all the atoms via the `atom` class. The atoms are stored in the array `m_atoms`, the system size is stored and the initial positions are stored in a separete variable `initialPosition` in the atom class to calculate the mean square displacement later.

The `step` function moves the particles to the next position by using the `integrator` class. Most of the time is spent in the `calculateForces` function in the `lennardJones` class that calculates and set the forces in the `atom` class. Then the `integrator` class moves the particles to a new position and the `applyPeriodicBoundaryConditions` checks if the particles have left the simulation box. If so they are moved to the other side of the box and the initial position is moved to be able to calculate the mean square displacement properly (as if there where no PBCs).When the particles have been moved one step the `statisticsSampler` class sample the statistical properties introduced in section 2.5 and at last the statistics is saved to a file to plot the data with a python script.

The program can also produce a file called `movie.xyz`[9] that can be used to visulize the simulation in a visualization program. In this project we have used `Ovito - The Open Visualization Tool`[10] to look at the system development in time.

3.3 Intergration methods

In the skeleton code the Euler-Cromer method is used to find the next timestep. In MD the Velocity Verlet integrator is usually used. This is because other integration algorithms tend to drift the energy over time, which we want to avoid[11]. The Velocity Verlet algorithm consists of three steps (in addition to computing the forces)

$$\mathbf{v}(t + \Delta t/2) = \mathbf{v}(t) + \frac{\mathbf{F}(t)}{m}\frac{\Delta t}{2} \tag{20}$$

$$\mathbf{r}(t + \Delta t) = \mathbf{r}(t) + \mathbf{v}(t + \Delta t/2)\Delta t \tag{21}$$

$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t + \Delta t/2) + \frac{\mathbf{F}(t + \Delta t)}{m}\frac{\Delta t}{2}. \tag{22}$$

To measure the drift of the energy we will use the standard deviation of the total energy E:

$$\sigma_E = \sqrt{\langle E^2\rangle - \langle E\rangle^2}$$

that we will calculate using the Euler-Cromer method and the Velocity-Verlet integrator and compare the results. We will also compare the execution time for both of the integrators.

3.4 Implementation of the `calculateForces` function

In section 3.3 we see that to calculate the next time step for the particles we need to calculate the forces. This is very time consuming because we have to sum over all pairs of particles to find the force for one particle (see section 2.5). Therefore a lot of time can be saved if the code for the calculation of forces is as efficient as possible. A better numerical expression for the force is

$$\begin{aligned}\mathbf{F}(r_{ij}) &= \frac{24\epsilon}{\sigma^2}\Big(\frac{\sigma}{r_{ij}}\Big)^2\Big(\frac{\sigma}{r_{ij}}\Big)^6\Big[2\Big(\frac{\sigma}{r_{ij}}\Big)^6 - 1\Big]\Big(x_{ij}\hat{\mathbf{i}} + y_{ij}\hat{\mathbf{j}} + z_{ij}\hat{\mathbf{k}}\Big)\\ &= 24\epsilon\sigma^6 r_{ij}^{-2} r_{ij}^{-6}\big(2\sigma^6 r_{ij}^{-6} - 1\big)\end{aligned}$$

and for the potential energy

$$U(r_{ij}) = 4\epsilon\sigma^6 r_{ij}^{-6}\big(\sigma^6 r_{ij}^{-6} - 1\big)$$

so that the powers of r_{ij} is calculated as few times as possible. With these expressions r_{ij}^{-6} can be calculated only once for each loop over the particle pairs.

Another smart way to make a more efficient code when calculating the forces is to realize that in the limit $r_{ij} \to \infty$ both the force and the potential energy is zero. In figure 2 the potential energy (red) and the force (blue) plotted against the distance between two atoms r_{ij}. We can see that the energy and the force reaches zero for large values of r_{ij}. The point for which the potential energy and the force is nearly zero, r_{cut}, is marked with a dot in the figure. A good approximation is to stop calculating forces and energies for distances larger than r_{cut} and set all forces and energies for $r_{ij} > r_{cut}$ to zero.

[9] Read more about the file format here: `https://en.wikipedia.org/wiki/XYZ_file_format`

[10] `http://www.ovito.org` 11.dec 19:30

[11] See for instance `https://en.wikipedia.org/wiki/Verlet_integration#Velocity_Verlet` about conserved quantities.

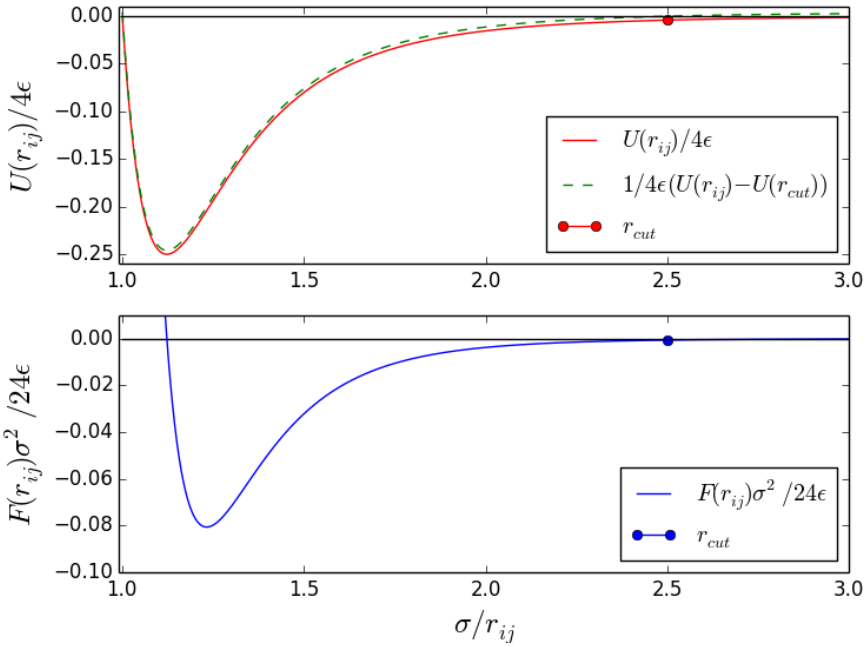


Figure 2: Top: The Lennard-Jones potential (red), or the potential energy $U(r_{ij})$ between two atoms i and j plotted against the distance between atom i and j, r_{ij}. The point $(r_{cut}, U(r_{cut}))$ is marked with a red dot. The shifted potential function is also plotted in green. **Bottom:** The force (blue) between two atoms i and j plotted against the distance between atom i and j, r_{ij}. The point $(r_{cut}, F(r_{cut}))$ is marked with a blue dot.

The implementation of r_{cut} will save the algorithm a lot of time and still be a good approximation since the force and the potential energy is esensially zero anyway. The only problem is that the potential energy function will be discontinuously at the point r_{cut} when the energy is set to zero after r_{cut}. A solution to this is to shift the potential function slightly so that the energy is zero at r_{cut} and still set to zero for $r_{ij} > r_{cut}$. The shifted potential function is marked with green in figure 2. This will only shift the values for the potential and total energy, but it will not affect the physics of the problem. The statistical properties of the system will not be affected by the shift in energy.

The minimum image convention

When an atom leave the simulation box it must re-enter the box on the opposite side, as described in section 2.3. An atom at the edge of the simulation box must also interact with an atom at the other side of the box 'through' the wall, or interact with the image of the simulation box. But this atom is also contained inside the box so the atoms interact 'twice', once through the wall and once from across the box. To avoid counting this interaction twice we will apply the minimum image convention. An atom will interact with the other atoms only once by choosing the image (or the simulation box) that gives the minimum distance[12] dx. A psudo-code is listed here:

```
if(periodic_x) then
  dx = x(j) - x(i)
  if (dx >   x_size * 0.5) dx = dx - x_size
  if (dx <= -x_size * 0.5) dx = dx + x_size
endif
```

where `x_size` is the lengt of the simulation box in x-direction. We do a cut-off, or choose the image of the particle when the distance is longer than the length of the simulation box.

4 Results

All unitless numbers are in the so called MD units listed in appendix A.

4.1 Verifying the implementation of the code by using Ovito and choice of parameters

To the left in figure 3 we see a screen shot of the initial frame of the simulation box with one unit cell from `Ovito`. We see that this is exactly the unit cell introduced in section 2.1 and figure 1. To the right in figure 3 we see a screen shot of the initial frame of the simulation box with 10 unit cells from `Ovito`, creating a cubic lattice. After looking at the initial frame of the simulation box in `Ovito` it seems safe to assume that the program produces the desired initial positions for the Argon atoms.

In the skeleton code the particles was unaffected by forces in the simulation. Before the Lennard-Jones potential and the calculation of the forces was implemented the particles was only given an initial speed by the maxwell boltzmann distrubution and should thus move in a straight line. In `Ovito` one could see that the particles moved in a straight line, and the

[12] `https://en.wikipedia.org/wiki/Periodic_boundary_conditions#Practical_implementation:_continuity_and_the_minimum_image_convention`

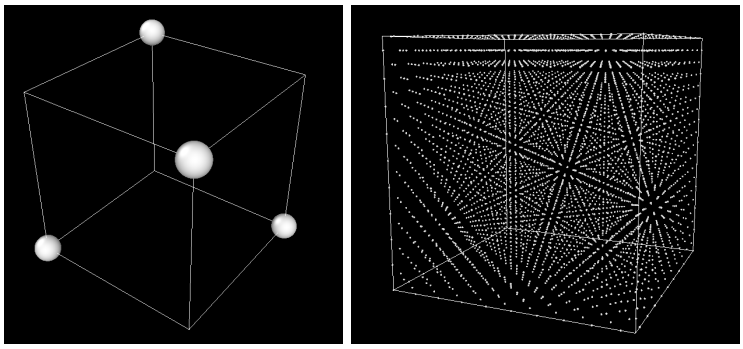


Figure 3: Left: A screen shot of the initial frame of the simulation box with one unit cell from `Ovito`. **Right:** A screen shot of the initial frame of the simulation box with 10 unit cells from `Ovito`, creating a cubic lattice.

simulation box expanded as the particels reached further out. After the periodic boundary conditions was implemented the simulation box maintained a constant size when looked at in `Ovito`. One could see that the particles still moved in a straight line, but they would jump to the oposite side of the box if they where to move out of the box.

After the implementation of the forces one could easy see (in `Ovito`) that the particles was contained inside the initial crystal structure, for instance at the temperature $T_i = 300$K. Just by looking at the simulation one could determine that the atoms formed a solid. At a high temperature, for instance $T_i = 1000K$ one could see that the atoms moved freely and that the crystal or solid most likely had melted. The implementation of the PBCs and the `calculateForces` function appears to be correct.

4.2 Comparing the two integrators

In the top plot in figure 4 we can see the standard deviation of the total energy σ_E plotted versus the timestep dt for the Euler-Cromer and Velocity Verlet integrator. The units on the axis are the MD units explained in appendix A. We see that the drift in the energy is significantly larger for the Euler-Cromer method than for the Velocity Verlet integrator and that it increases with increasing dt for the Euler-Cromer method. For the Velocity Verlet integrator the choice of dt does not matter much, the energy drift, or the value for σ_E is small for all values of dt. In the further calculations we have therefore chosen dt=0.05.

In the bottom plot in figure 4 we can see the execution time plotted versus the timestep for the Euler-Cromer and Velocity Verlet integrator. We see that there is not much difference in the execution time between the two integrators and that the dependence on the value of dt seems rather random.

Both plots in figure 4 where obtained with 5 unit cells in each direction, initial temperature $T_i = 2.5 \approx 300$ K, lattice constant $b = 5.26$Å and 1000 timesteps in the simulation. Similar plots where obtained when running with 10 unit cells in each direction.

In figure 5 we see the execution time plotted versus the number of unit cells in each direction for the Euler-Cromer and Velocity Verlet integrator. We see that for small number of unit cells the execution time is about the same for both integrators, they are equally fast. For 15 unit cells in each direction the two integrators deviate slightly in the execution time, but the execution time is quite large, $t_{exe} \approx 800$s ≈ 13min so that the deviation in the execution

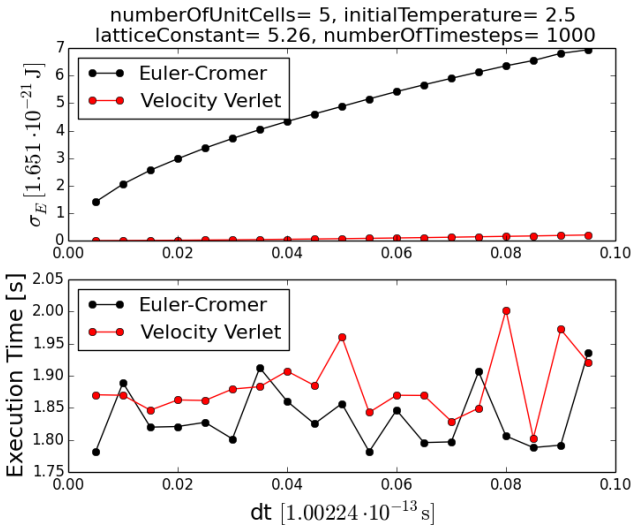


Figure 4: Top: The standard deviation of the total energy σ_E plotted versus the timestep dt for the Euler-Cromer and Velocity Verlet integrator with 5 unit cells in each direction, initial temperature $T_i = 2.5 \approx 300$ K, lattice constant $b = 5.26$Å and 1000 timesteps in the simulation. **Bottom:** The execution time plotted versus the the timestep dt for the Euler-Cromer and Velocity Verlet integrator with 5 unit cells in each direction, initial temperature $T_i = 2.5 \approx 300$ K, lattice constant $b = 5.26$Å and 1000 timesteps in the simulation.

time between the integrators are small compared to the execution time. Thus the choice of integrator does not affect the execution time much and we should choose the integrator that gives the best physical results. The best physical results are when the drift in the energy or σ_E is as small as possible, we should therefore use the Velocity Verlet integrator. The further calculations are done with the Velocity Verlet integrator.

4.3 Equilibium time and energy conservation

In figure 6 we see the energy plotted versus the time with initial temperature $T_i = 2.5 \approx 300$ K for two different number of unit cells in each direction. We see that the total energy is conserved, or constant and thus the implementation of r_{cut} appeares to be correct. This coresponds well with the low value of σ_E found in section 4.2 (figure 4). We can see that the system is initialized with high kinetic energy and low potential energy and that the two interchanges energy giving that the total energy is conserved. The system stabilizes after a short period of time, it reaches an equilibrium state when the kinetic and potential energy fluctuates around a constant energy. The equilibrium state is reached at least for $t = 10$, or after 20 timesteps.

The kinetic energy drops below the equilibrium value before it stabilizes. This is because the particles are all initialized with different velocities and move until they are stoped, at about the same time, by the forces of the surrounding particles before they find an equilibrium position and oscillates around this position.

The top plot in figure 6 is obtained with 5 unit cells in each direction and the bottom plot is obtained with 10 unit cells in each direction. We see that the plot with only 5 unit cells fluctuates more after equilibrium than the plot with 10 unit cells.

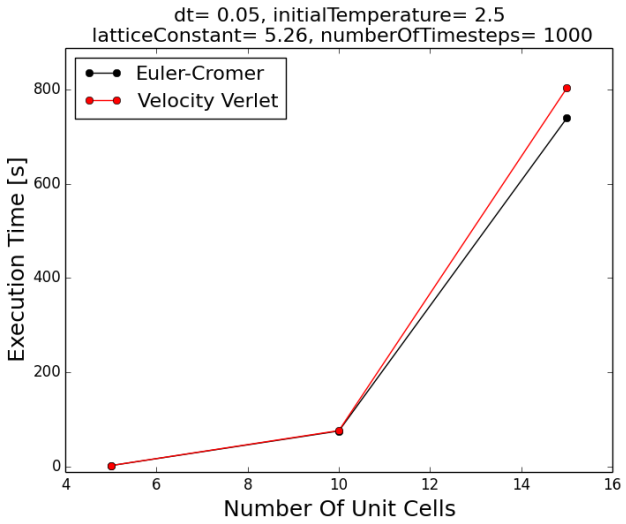


Figure 5: The execution time plotted versus the number of unit cells in each direction for the Euler-Cromer and Velocity Verlet integrator with initial temperature $T_i = 2.5 \approx 300$ K, lattice constant $b = 5.26$Å, $dt = 0.05 \approx 5 \cdot 10^{-15}$s and 1000 timesteps in the simulation.

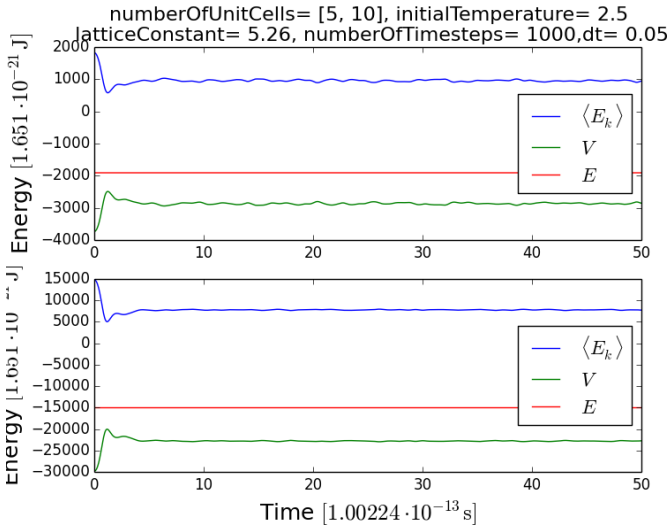


Figure 6: The kinetic (blue), potential (green) and total (red) energy plotted versus the time with initial temperature $T_i = 2.5 \approx 300$ K, lattice constant $b = 5.26$Å, $dt = 0.05 \approx 5 \cdot 10^{-15}$s and 1000 timesteps in the simulation. **Top:** 5 unit cells in each direction. **Bottom:** 10 unit cells in each direction.

In figure 7 we also see the energy plotted versus the time, but with the initial tempererature $T_i = 8.35 \approx 1000$ K. The initial temperature is a lot larger than in figure 6 so the system have a higher initial kinetic and potential energy giving a higher total energy. The total energy is still conserved but a larger amount of the energy now comes from the kinetic energy than in figure 6. The time the system uses to reach equilibrium is about the same as for $T_i = 2.5$. We notice that the fluctiations after reaching equilibrium is larger for 5 unit cells (top) than for 10 unit cells (bottom) in figure 7. The flutuations are even larger for $T = 8.35$ than for $T = 2.5$.

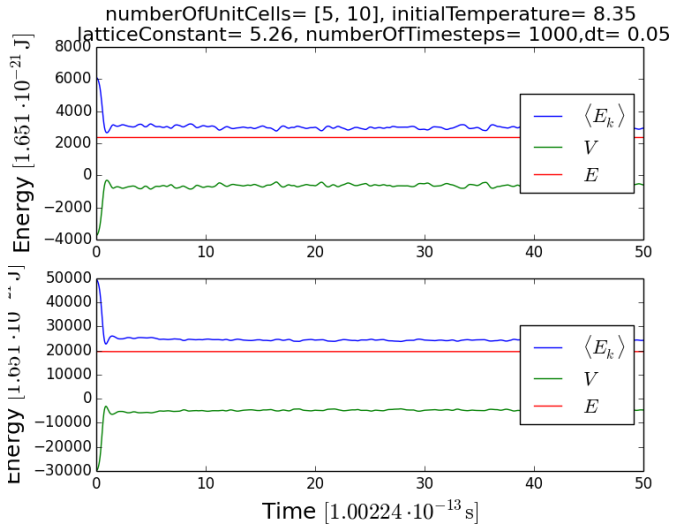


Figure 7: The kinetic (blue), potential (green) and total (red) energy plotted versus the time with initial temperature $T_i = 8.35 \approx 1000$ K, lattice constant $b = 5.26$Å, $dt = 0.05 \approx 5 \cdot 10^{-15}$s and 1000 timesteps in the simulation. **Top:** 5 unit cells in each direction. **Bottom:** 10 unit cells in each direction.

In figure 8 we see the instant temperature T plotted versus the time with initial temperature $T_i = 2.5 \approx 300$ K. Right after initializing the temperature drops quickly below the equilibrium value. This behaviour can also be observed in the kinetic energy in figure 6 and 7. Since the instant temperature is proportional to the kinetic energy (see section 2.5) it is expected that the two curves have the same behaviour.

The value for the instant temperature stabilizes quicly and oscillates around a constant value, just as in the energy plots. It is therefore safe to assume that the system has reached equilibrium after at least 100 timesteps. In the following calculations we have used 10000 timesteps which is long after the system have reached equilibrium.

The fluctuations in the temperature is, as for the energy plot, larger for the plot with 5 unit cells in each direction than with 10 unit cells. The fluctuations are even larger in figure 9 where the instant temperature T plotted versus the time with initial temperature $T_i = 8.35 \approx 1000$ K. It apeares as if the system is more unstable and 'noisy' for large temperatures and few unit cells. Even though the trend is that the data obtained with a larger number of unit cells is more stable than with few unit cells, we have chosen to use 5 unit cells in each direction in the following calculations. This is to save time, the execution time for 10 unit cells are quite long compared with the execution time for 5 unit cells.

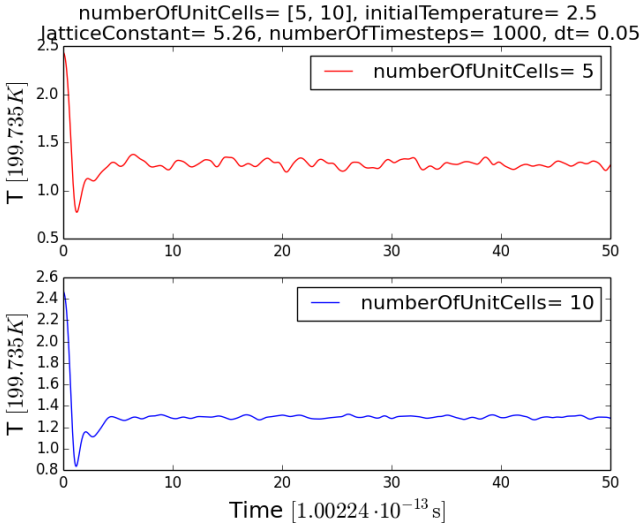


Figure 8: The instant temperature T plotted versus the time with initial temperature $T_i = 2.5 \approx 300$ K, lattice constant $b = 5.26$Å, $dt = 0.05 \approx 5 \cdot 10^{-15}$s and 1000 timesteps in the simulation. **Top:** 5 unit cells in each direction. **Bottom:** 10 unit cells in each direction.

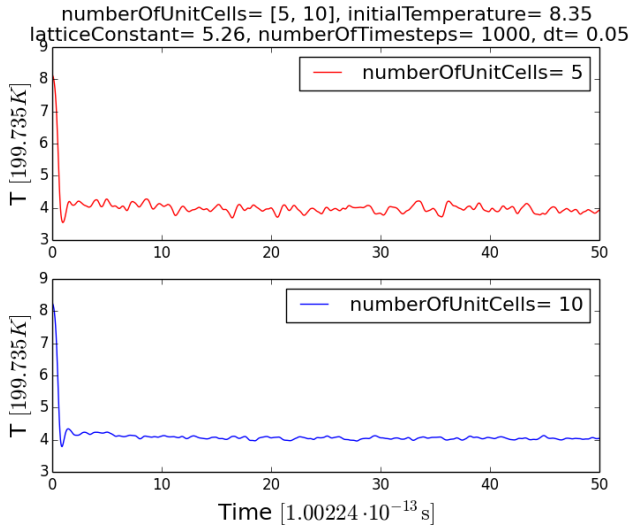


Figure 9: The instant temperature T plotted versus the time with initial temperature $T_i = 8.35 \approx 1000$ K, lattice constant $b = 5.26$Å, $dt = 0.05 \approx 5 \cdot 10^{-15}$s and 1000 timesteps in the simulation. **Top:** 5 unit cells in each direction. **Bottom:** 10 unit cells in each direction.

The temperature ratio

In figure 10 we see the ratio between the instant temperature T and the initial temperature T_i plotted versus the time for the initial temperatures $T_i \in [0.5, 18.5]$. We see that the eqiulibrium state is reached for all of the temperatures after the time $t = 10$, as also seen in the energy plots. The ratio T/T_i is not equal for all of the initial temperatures, but it does center around the value 0.5 so that:

$$\begin{aligned} T/T_i &\approx \frac{1}{2} \\ \Rightarrow T &\approx \frac{T_i}{2} \end{aligned} \tag{23}$$

The initial value for the temperature will approximately drop to half after the equilibrium state is reached.

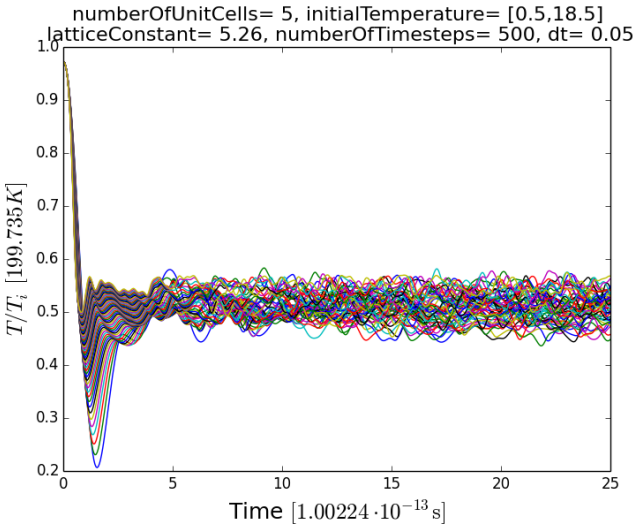


Figure 10: The ratio between the instant temperature T and the initial temperature T_i plotted versus the time (first 500 timesteps) for initial temperatures $T_i \in [0.5, 18.5]$, lattice constant $b = 5.26$Å and $dt = 0.05 \approx 5 \cdot 10^{-15}$.

4.4 The mean square displacement and the diffusion constant

In figure 11 we see the diffusion constant (top) and the mean square displacement (bottom) for the particles plotted versus the time for initial temperatures $T_i \in [3.5, 6.5]$. We can see that the diffusion constant diverges against a constant value for large values for the time. The highest temperatures have the largest values for all temperatures, but it is difficult to tell the different initial temperatures apart by looking at this plot. After 10000 timesteps the diffusion constant have still not stabilized entirely, but we will take the diffusion constant after 10000 timesteps to be the diffusion constant after equilibrium D_{eq}.

It is easier to tell the initial temperatures apart in the bottom plot in figure 11. We can see that the mean square displacement of the particles is zero for the low temperatures and that it for higher temperatures starts to grow for larger values of the time. When the system is in a solid state it is expected that the diffusion constant is be zero, or that $\langle r^2(t) \rangle$ is zero. For

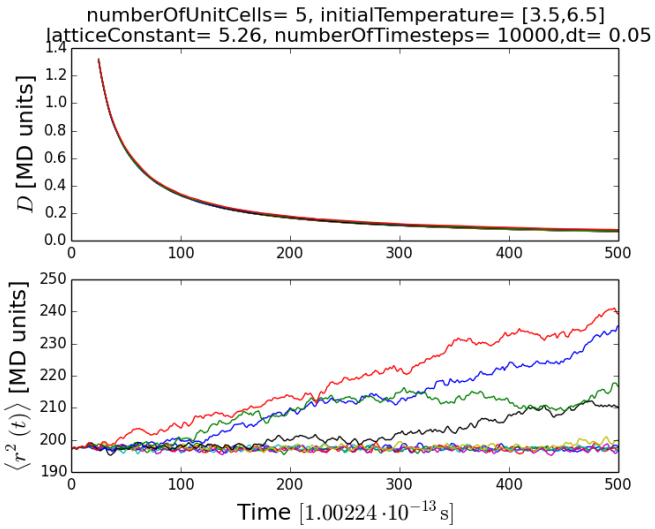


Figure 11: Top: The diffusion constant D plotted versus the time with 10000 timesteps for initial temperatures $T_i \in [0.5, 18.5]$. **Bottom:** The mean square displacement $\langle r^2(t) \rangle$ of the particles plotted versus the time with 10000 timesteps for initial temperatures $T_i \in [0.5, 18.5]$. The units are the MD units listed in appendix A.

higher temperatures the diffusion constant would be non-zero giving the slope seen for higher temperatures in the $\langle r^2(t) \rangle$ plot. It is hard to tell eqactly for which instant temperature the system melts, but it must be in the range with initial temperatures $T_i \in [3.5, 6.5]$.

In figure 12 we see the temperature ratio T_{eq}/T_i (top) and the diffusion constant (bottom) after reaching equilibrium D_{eq} plotted versus the instant temperature T_{eq} whereT_{eq} is the instant temperature after 10000 timesteps. We see that the diffusion constant is zero for all temperatures T_{eq} up to the point $T = 2.3 \approx 275.4$K where it suddenly grows. It apeares as if the gowth is quite linear, but there is a lot of noise in the datapoints. We can also see that the 'lines cross' at one point for high temperatures. In the top plot in figure 12 we see that the temperature ratio for low temperatures is approximately 0.5 and follows equation 23. For the temperature $T = 2.3 \approx 275.4$K we see that the ratio suddenly drops to 0.4 before it starts growing again.

It appeares to be a lot of noise in the datapoints in both plots in figure 12 as well, at least for high temperatures. This might be explained in the fact that we have only used 5 unit cells in each direction. We saw the same 'noise' effects in the energy plots that apeared to be due to the small system size.

When looking in `Ovito` one can try to see if the mean square displacement appeares to be small for temperatures below and high above $T = 2.3$. One can sligtly see a change in the movement of the particles around $T = 2.3$, but it is hard to tell a change exactly with the eyes. The following was observed in `Ovito`:

- $T = 2.2$: The atoms are containes in a crystal structure. The atoms moves quite a lot, but it apeares as if none of the atoms 'escapes' the crystal structure. The crystal appeares to be contained and the system a solid.

- $T = 2.5$: The atoms moves a lot, but one can see vaguely a 'left over' from the crystal structure. The atoms seems to be contained inside an area, and a few atoms 'escape'

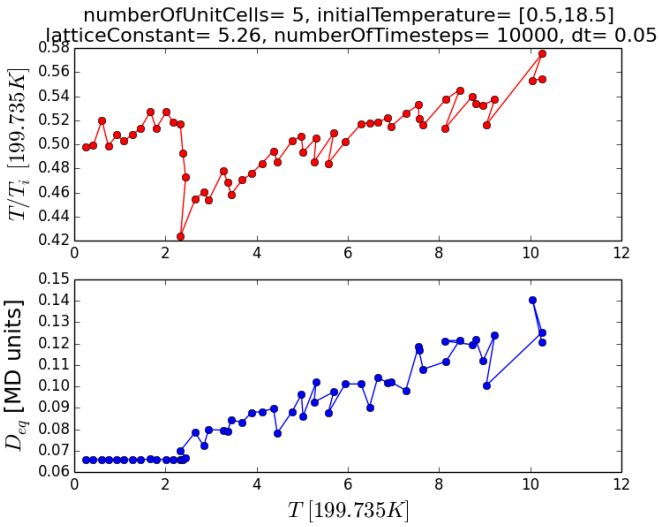


Figure 12: Top: The temperature ratio $T/T - i$ plotted versus the instant temperature T with 10000 timesteps. **Bottom:** The diffusion constant after reaching equilibrium D_{eq} plotted versus the instant temperature T with 10000 timesteps. The units are the MD units listed in appendix A.

from the structure giving defects in the crystal. It appeares as if the solid has melted.

5 Conclusions and further perspectives

In figure 12 we clearly see discontinuously points in both the diffusion constant and in the temperature ratio. We see that the diffusion constant is zero for low themperatures, but starts to grow at $T = 275$ K. The melting temperature of the Argon crystal must therefore be at $T = 275$ K. This is also consistent with the observations in `Ovito` and it also seems reasonable that there should be a discontinuity for one parameter at the time of a phase transition as the one seen in figure 12.

The next step would have been to compare this melting temperature with experimental data for Argon. Most likely one would have to adjust the density in the simulation to match the density in the experimental setup. This is easily done by adjusting the lattice constant b because the pressure is proportional to the inverse of the lattice constant cubed. If one where to run the simulations again for a different lattice constant it would have been interesting to run the simulation with a larger number of unit cells to see if this would affect the uncertainties, or noise in the results.

Appendices

A Units

The program uses by default from the skeleton code a set of units that we will call MD units.The chosen units are the following:

$$\begin{aligned} 1 \text{ unit of mass } &= 1 \text{ a.m.u} = 1.661 \times 10^{-27} \text{ kg}, && (24)\\ 1 \text{ unit of length } &= 1.0 \text{ Å} = 1.0 \times 10^{-10} \text{ m}, && (25)\\ 1 \text{ unit of energy } &= 1.651 \times 10^{-21} \text{ J}, && (26)\\ 1 \text{ unit of temperature} &= 119.735 \text{ K}. && (27) \end{aligned}$$

Boltzmann's constant is then equal to one, and other units can be derived by using known relations. We can for example find the unit of time by using $E = mc^2$. We have that

$$\text{Energy} = \text{Mass} \times \frac{\text{Length}^2}{\text{Time}^2}, \tag{28}$$

which can be solved for time

$$\text{Time} = \text{Length} \times \sqrt{\frac{\text{Mass}}{\text{Energy}}}. \tag{29}$$

By inserting the values above, we get

$$1 \text{ unit of time } = 1.0 \times 10^{-10} \sqrt{\frac{1.661 \times 10^{-27}}{1.651 \times 10^{-21}}} \text{ s} = 1.00224 \times 10^{-13} \text{ s}. \tag{30}$$

This way, we can continue working out the values of the other units. In the skeleton code, the `unitConverter` class calculates transitions between the SI units and the MD units.