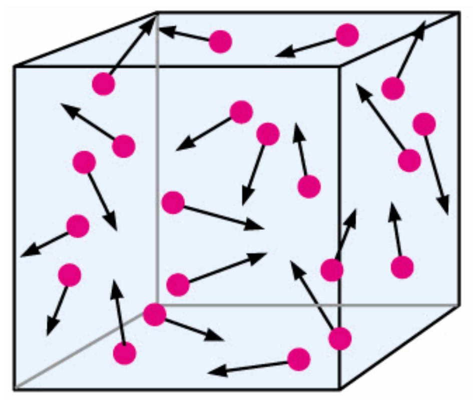
A study of the properties of argon using Molecular Dynamics

Candidate 21 in cooperation with Candidate 11

December 8, 2014



Link to github: Source Code

1 Introduction

In this project we will implement a model called Molecular Dynamics (MD). MD allows us to study the dynamics of atoms and explore the phase space. The atoms interact through a force given as the negative gradient of a potential function. In this project we use the Lennard-Jones potential. With this force we can integrate Newton's laws. While exploring the phase space, we will sample statistical properties such as energy, temperature, pressure and heat capacity. Further we will look at phase transitions. Specifically we apply the model to argon in an infinite box.

Initially we were given a C++ code skeleton containing the basic features of an MD code. The program creates 100 argon atoms and place them uniformally inside a box of 10 Ångstroms (Å). Each atom is given a velocity according to the Maxwell-Boltzmann distribution. The program evolves the system in time with no forces so that all atoms move in a straight line. It creates a file called movie.xyz containing all timesteps so it can be visualized with Ovito. In the following we will not discuss the details of the code, but where there are key concepts we present the algorithm in pseudo-code. All figures are created with python scripts contained in the above github link.

2 Setting up the system

Before we can begin to do calculations, we have to set up the physical system which we will consider in a reasonable way. We first introduce specific MD units and give a short description of the structure of the code. Instead of placing the atoms uniformally in space, we introduce the FCC-lattice and we implement periodic boundary conditions. This enables us to simulate an infinite system.

2.1 Units

Since we are working with sizes on an atomic scale, it is convenient to introduce another set of units such that

$$1 \text{ unit of mass} = 1.661 \times 10^{-27}\text{kg}, \tag{1}$$
$$1 \text{ unit of length} = 1\text{Å} = 1.0 \times 10^{-10}\text{m}, \tag{2}$$
$$1 \text{ unit of energy} = 1.651 \times 10^{-21}\text{J}, \tag{3}$$
$$1 \text{ unit of temperature} = 119.735\text{K}. \tag{4}$$

With this set of units the Boltzmann constant is equal to one.

$$\begin{aligned} k_B &= 1.381 \times 10^{-23}\frac{J}{K} \\ &= \frac{1.381 \cdot 119.375}{1.651 \cdot 100}\,[\text{energy/temperature}] \\ &= 1\,[\text{energy/temperature}]\,. \end{aligned}$$

Other relevant units can be derived in a similiar way using known relations. In the code we were given, there is a fully functional class, *unitConverter*, that converts units from SI-units to our chosen set of units.

2.2 Code structure

We have chosen to write the program in C++ and the code is object-oriented. In addition to the *main* function, we have the following classes in the program.

- unitConverter
- system
- statisticsSampler
- io
- CellList
- BerendsenThermostat
- atom
- potential
- lennardjones
- vec3
- random
- velocityverlet
- eulercromer
- integrator

The classes *unitConverter, vec3, random and eulercromer*, which all were present in the code skeleton, is left unchanged throughout the project. The classes *BerendsenThermostat*, *CellList* and *velocityverlet* has been written from scratch and some of its features will be discussed later in the text. For the remaining classes we have done modifications where it has been necessary and some of these changes will be highlighted.

An instance of the class *system* contains all information about the system under consideration, such as the position and velocity of atoms, while the other classes are either used to perform operations on the system or data handling. In order to perform more complicated simulations we have written a python framework, which is provided in the above github-link.

2.3 Periodic Boundary Conditions

The first modification we do to the code is that we want to simulate a system of infinite size, eliminating boundary effects. In order to do so we apply periodic boundary conditions. We restrict ourselves to a cubic system with L being the length, width and height of the box. Periodic boundary conditions ensures that if an atom leaves the box in any direction, we give it a position keeping it in the box. It is implemented in the function *applyPeriodicBoundaryConditions* in the *System* class. For the x-direction we have the algorithm 1, which is done exactly the same way in the y and z-direction.

```
1 Assume that $r_x$ is the position of the atom in the $x$-direction;
2 if $r_x < 0$ then
3     $r_x = r_x + L$;
4 else if $r_x \geq L$ then
5     $r_x = r_x - L$
6 end
```

Algorithm 1: Periodic Boundary Conditions

We work with argon which is a noble gas. Noble gases are nearly ideal gases under standard conditions, so we give the atoms in the gas velocities according to the Maxwell-Boltzmann distrubution since it describes particle speeds in idealized gases. This results in a non-zero net momentum in the system. We want to have zero net momentum because we want the center of mass of the system to be stationary. We do this by calculating the systems net-momentum, p^{net}, and the net-momentum per atom

$$p_{atom} = \frac{p^{net}}{N_{atoms}}.$$

Then we can adjust the velocities of the atoms,

$$v_i^{new} = v_i^{prev} - \frac{p_{atom}}{m},$$

which results in zero net-momentum in the system. This is done in the function *removeMomentum* in the *System* class.

2.4 FCC

At this point the atoms are uniformly distributed in space. This is not very physical so we want to place the atoms in a crystal structure. The Face-centered cubic (FCC) lattice is the crystalline structure of argon. A lattice is built up by *unit cells* -a group of atoms- so that the larger system can be created by repeating these cells in space. An FCC lattice unit-cell of size bÅ, with b being the lattice constant, consists of four atoms with local coordinates.

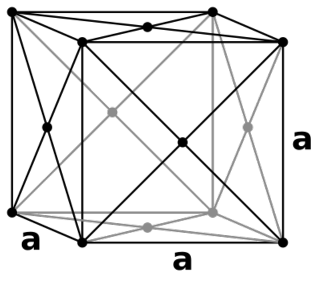
$$\mathbf{r}_1 = 0\hat{\mathbf{i}} + 0\hat{\mathbf{j}} + 0\hat{\mathbf{k}}, \tag{5}$$

$$\mathbf{r}_2 = \frac{b}{2}\hat{\mathbf{i}} + \frac{b}{2}\hat{\mathbf{j}} + 0\hat{\mathbf{k}}, \tag{6}$$

$$\mathbf{r}_3 = 0\hat{\mathbf{i}} + \frac{b}{2}\hat{\mathbf{j}} + \frac{b}{2}\hat{\mathbf{k}}, \tag{7}$$

$$\mathbf{r}_4 = \frac{b}{2}\hat{\mathbf{i}} + 0\hat{\mathbf{j}} + \frac{b}{2}\hat{\mathbf{k}}. \tag{8}$$

Figure 1: FCC Unit Cell



We let N_{atoms} denote the number of atoms in the system and V the volume. We restrict ourselves to the case where the number of unit cells are the same in every dimension, $N_x = N_y = N_z = N_{cells}$. Since the atoms now are placed in an FCC lattice, $N_{atoms} = 4N_{cells}^3$, and $V = (N_{cells}b)^3$. Then the number density

$$\rho = \frac{N_{atoms}}{V}$$

is given by

$$\rho = \frac{4N_{cells}^3}{(N_{cells}b)^3} = \frac{4}{b^3}.$$

Instances of the class *atom* creates an atom and we place the atoms in the FCC-lattice using algorithm 2, implemented in the function *createFCCLattice* in the *System* class.

```
$N \sim$ number of unit-cells and $b \sim$ lattice constant;
$r_1, r_2, r_3, r_4$ is the position of the four local atoms;
for $i = 0 : N$ do
    for $j = 0 : N$ do
        for $k = 0 : N$ do
            Create four atoms;
            $r_1 = (i * b, j * b, k * b)$;
            $r_2 = (b * (0.5 + i), b * (0.5 + j), b * k)$;
            $r_3 = (i * b, b * (0.5 + j), b * (0.5 + k))$;
            $r_4 = (b * (0.5 + i), j * b, (k + 0.5) * b)$;
        end
    end
end
```

Algorithm 2: FCC

3 Implementing the physics

Now that the setup of the system is complete, we want to add interactions between particles. This is done by using the Lennard-Jones (LJ) potential. We calculate the forces between atoms given by the LJ potential and introduce the Velocity Verlet scheme instead of the Euler Cromer method. We measure the kinetic and potential energy in the system and estimates its temperature. Further we investigate the computaional efficiency of the program and introduce the notion of cell lists to improve the efficiency. Also we implement the Berendsen thermostat in order to control the temperature of the system.

3.1 The Velocity Verlet scheme

Initially there is not implemented any interactions between the atoms and we want to add this to the code. We use the the LJ potential which is given by

$$U(r_{ij}) = 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] \tag{9}$$

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 and σ is the distance at which the potential is zero. The force is given as the negative gradient of the potential,

$$\mathbf{F}(r_{ij}) = -\nabla U(r_{ij}) = -\frac{\partial U}{\partial r_{ij}} \frac{x_{ij}}{r_{ij}} \hat{\mathbf{i}} = -\frac{\partial U}{\partial r_{ij}} \frac{y_{ij}}{r_{ij}} \hat{\mathbf{j}} = -\frac{\partial U}{\partial r_{ij}} \frac{z_{ij}}{r_{ij}} \hat{\mathbf{k}}. \tag{10}$$

We calculate $\partial U / \partial r_{ij}$.

$$\begin{aligned}\frac{\partial U}{\partial r_{ij}} &= 4\epsilon\frac{\partial}{\partial r_{ij}}\left(\sigma^{12}r_{ij}^{-12} - \sigma^6 r_{ij}^{-6}\right) && (11)\\ &= 4\epsilon(-12\sigma^{12}r_{ij}^{-13} + 6\sigma^6 r_{ij}^{-7}) && (12)\\ &= \frac{4\epsilon}{r_{ij}}\left[6\left(\frac{\sigma}{r_{ij}}\right)^6 - 12\left(\frac{\sigma}{r_{ij}}\right)^{12}\right] && (13)\\ &= \frac{24\epsilon}{r_{ij}}\left[\left(\frac{\sigma}{r_{ij}}\right)^6 - 2\left(\frac{\sigma}{r_{ij}}\right)^{12}\right]. && (14)\end{aligned}$$

Then we have

$$\mathbf{F} = \frac{24\epsilon}{r_{ij}}\left[2\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^6\right]\left(\frac{x_{ij}\hat{\mathbf{i}} + y_{ij}\hat{\mathbf{j}} + z_{ij}\hat{\mathbf{k}}}{r_{ij}}\right).$$

When we implement this force we have to remember that we use periodic boundary conditions and that the positions of the atoms must satisfy the so called minimum image criterion. Since we simulate an infinite system, this ensures that we choose the shortest distance available. If $|r_{ij}| > L/2$ we adjust the positions in the following way.

```
1 Assume that $r_{ij}$ is the position between two atoms $i$ and $j$;
2 if $r_{ij} > L/2$ then
3     $r_{ij}^{new} = r_{ij} - L/2$;
4 else if $r_{ij} < -L/2$ then
5     $r_{ij}^{new} = r_{ij} + L/2$
6 end
```

Algorithm 3: Minimum Image Criterion

The force goverened by the LJ potential combined with the minimum image criterion is implemented in the function *calculateForces* in the *LennardJones* class.

With the force at hand, we now want to evolve the system in time. Since we have no other forces than the one given by the LJ potential, we can obtain the new velocities and positions of an atom pair i, j by integrating newtons second law,

$$m_{argon}\frac{dv_i}{dt} = F(r_{ij})$$

and using newtons third law

$$m_{argon}\frac{dv_j}{dt} = -F(r_{ij}).$$

In the code skeleton the Euler-Cromer method is used for integrating Newtons laws. However, in MD it is important that the total energy is conserved. The

Velocity Verlet scheme given by

$$\begin{aligned} v(t+\Delta t/2) &= v(t) + \frac{F(t)}{m}\frac{\Delta t}{2}, \\ r(t+\Delta t) &= r(t) + v(t+\Delta t/2)\Delta t, \\ v(t+\Delta t) &= v(t+\Delta t/2) + \frac{F(t+\Delta t)}{m}\frac{\Delta t}{2}, \end{aligned}$$

conserves the energy better than the Euler-Cromer method. From Jensen [1] we have that the global error of the Velocity Verlet method is $O(\Delta t^4)$ in position and $O(\Delta t^2)$ in velocity, while Euler-Cromer has global error $O(\Delta t^2)$ in both position and velocity. The error in the potenial energy, given by the LJ potenial, will be smaller using Velocity Verlet instead of Euler-Cromer since it calculates the position more accurately. Hence the total energy will be closer to its exact value assuming that

$$E_{tot} = E_k + E_p = \frac{1}{2}mv^2 + U_{LJ}.$$

The Velocity Verlet scheme is implemented in the *velocityverlet* class.

We can now run a simulation and visualize the results in Ovito. What we see is that if $N_{cells} = 5, T \leq 600$ the system is in a solid state and for greater temperatures the crystal is melting.

3.2 Sampling

Having implemented interactions between atoms, we now have a working MD code and we want to calculate some physical properties of the system. We sample the kinetic energy using

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

with m_i being the mass of each individual atom and v_i its velocity. Since we only have argon atoms we have $m_i = m_{argon} = 6.634 \cdot 10^{-26}$kg. The potential energy of the system is given by

$$V = \sum_{i<j} U(r_{ij}).$$

From the equipartition theorem we have

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

which we can use to define an instantaneous temperature

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

Because we look at an isolated system the total energy should be conserved. We want to choose Δt so that the fluctuations in the total energy are small. We tested different Δt values and found that if $\Delta t < 10^{-14}$ the results are

Table 1: $E_{tot}(\Delta t = 10^{-14})$ to the left and $E_{tot}(\Delta t = 10^{-15})$ to the right.

timestep	E_{tot}
0	-3036.0898
100	-3035.1226
200	-3035.0189
300	-3035.1919
400	-3035.2283

timestep	E_{tot}
0	-3036.0362
100	-3036.0136
200	-3036.0299
300	-3036.0251
400	-3036.0271

numerically unstable and physically irrelevant. Below we have listed the total energy for $\Delta t = 10^{-14}$ and $\Delta t = 10^{-15}$.

We see that for $\Delta t = 10^{-14}$ the fluctuations occur in the fourth significant digit, while for $\Delta t = 10^{-15}$ we have fluctuations in the sixth significant digit. We see that the fluctuations are smaller for smaller Δt.

When we look at the evolution of the system over time we can see that the potential energy starts at its minimum and rises before the system reaches equilibrium.

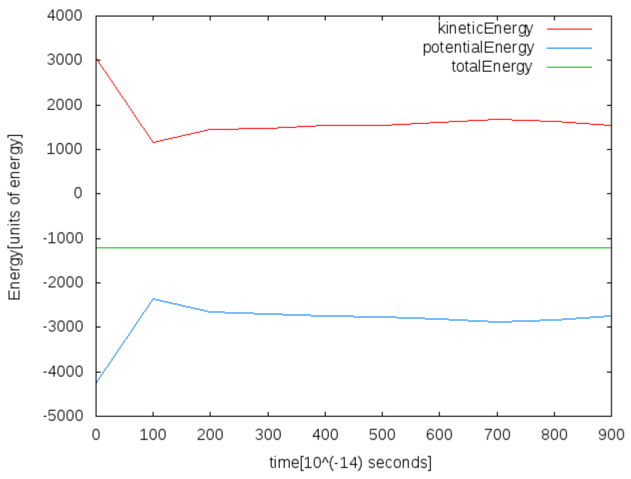


Figure 2: Energy plot

Since the total energy is conserved and the potential energy starts at its minimum, the kinetic energy has to start at its maximum and go down when the potential energy goes up. We can see from figure 2 that the systems kinetic energy is half of its initial value when the system is in equilibrium. From the relation

$$T = \frac{2}{3}\frac{E_k}{N_{atoms}k_B},$$

we expect that the temperature also have to fall to half of its initial value if we keep the number of atoms constant.

[1]Lectures2014 Computational Physics; Chapter 8

The second law of thermodynamics says [2]

> Any large system in equilibrium will be found in the macrostate with the greatest entropy (aside from fluctuations that are too small to measure).

This means that if we increase the number of atoms we expect the fluctuations to become smaller. In figure 3 we can see that this is the case.

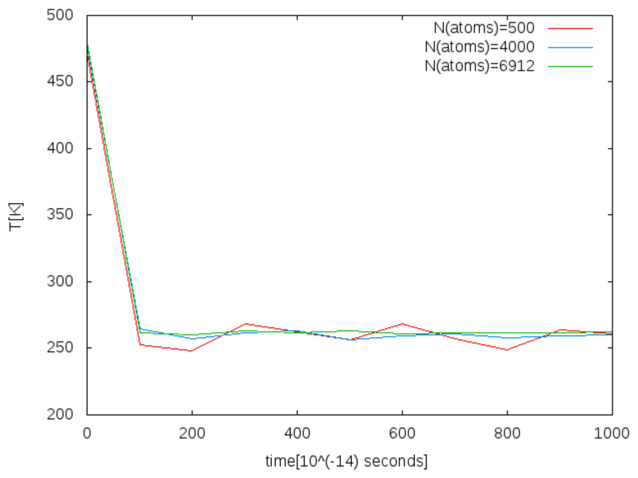


Figure 3: Temperature plot

[2] An introduction to Thermal Physics, First Edition, Daniel V. Schroeder

$Natoms$	runtime[seconds]
108	0.30
256	2.01
500	7.52
864	22.11
1372	55.30
2048	122.34

Table 2: runtime(N_{atoms})

3.3 Cell Lists

Having implemented the velocity verlet scheme and the LJ-potential we now want to investigate the efficiency of the code.

From table 2 we see that if we double the number of atoms the runtime increases approximately by a factor of four, which indicates that

$$\text{runtime}(N_{atoms}) = O(N_{atoms}^2).$$

This is too slow and we want to improve the runtime.

Right now the program calculates the forces between every atom pair. However, if the atoms are far apart the interactions between them are so weak that we can ignore it without loss of precision. Hopefully this will make the program faster. What we can do is to divide the physical space into boxes of size r_{cut} where r_{cut} is the distance at which the force between the atoms is negligible for all practical purposes.

We have to find the distance r_{cut}. For argon, optimal values of parameters in the LJ-potential are

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

Below we plot the magnitude of the force between two atoms, given by

$$\left|\frac{\mathbf{F}(r_{ij})}{\epsilon}\right| = \frac{24}{r_{ij}}\left[2\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right], \quad r_{ij} \in [2^{\frac{1}{6}}\sigma, 7\sigma].$$

We want to choose r_{cut} such that the code runs as fast as possible without losing too much precicision in the force. From figure 4 we see that when $r_{ij} = 10$ the force is approximately zero compared to its maximum value. By a closer examination we find that

$$\frac{F_{max}}{50} \approx |F(r_{ij} = 2.5\sigma)|$$

and we choose $r_{cut} = 2.5\sigma \approx 8.5$. The potential energy function now becomes discontinuous, but we can make it continuous by defining the truncated potenital energy,

$$U_{\text{trunc}}(r_{ij}) = \begin{cases} U(r_{ij}) - U(r_{\text{cut}}) & \text{for } r \leq r_{\text{cut}} \\ 0 & \text{for } r > r_{\text{cut}}. \end{cases} \tag{15}$$

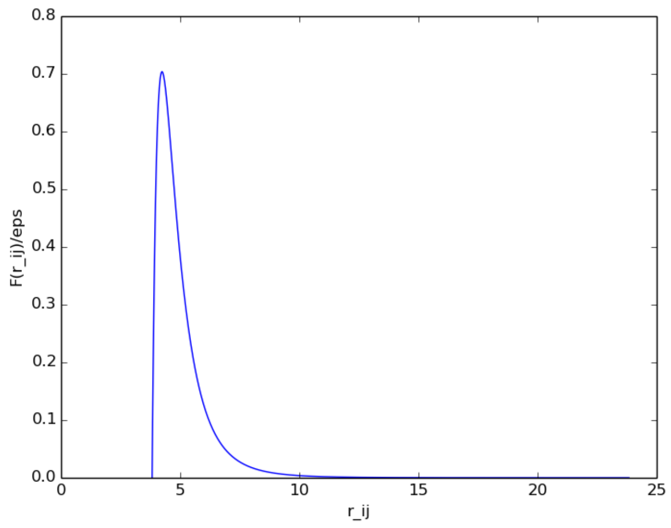


Figure 4: Force plot

In order to implement cell lists we make a class, *CellLists*, which divides the system into boxes of size r_{cut}. The number of boxes in each dimension is given by

$$n = \frac{N_{cells}}{r_{cut}}.$$

We have to keep track of which box each the atoms should be put into. This is achieved by the following algorithm

```
1 Let $r_{a,x}, r_{a,y}, r_{a,z}$ be the position of atom $a$ in the $x, y$ and
  $z$-direction;
2 Let cells[i,j,k] be a three-dimensional array containing all cells;
3 for $a = 0 : N_{atoms}$ do
4     pick atom number a;
5     i = $(r_{n,x}/N_{cells})n$;
6     j = $(r_{n,y}/N_{cells})n$;
7     k = $(r_{n,z}/N_{cells})n$;
8     Put atom a into cells[i,j,k].
9 end
```

which have to be done every time we update the positions and velocities of the atoms. Then for an atom a, we only have to compute interactions with atoms residing in the cells neighbouring and those atoms inside its own cell. In three-dimensions this corresponds to visiting 27 cells. We have implemented this in the function *calculateForcesWithCellLists* in the *LennardJones* class.

$Natoms$	runtime[seconds]
1372	7.822
2048	15.723
2916	17.5264
4000	20.5947

timestep	E_{tot}
0	-5134.4285
100	-5133.9839
200	-5134.3179
300	-5134.2957

Table 3: Runtime and energy conservation with cell lists

Now we want to check that the efficiency with this modification actually is better and we also have to make sure that the total energy of the system is still conserved. We can see from table 3 that if we double the number of atoms the runtime goes up by a factor of two in the worst case and even better at some points. This indicates that the runtime is of $O(N_{atoms})$, which is a vast improvement to what we had before. We measure the energy for $\Delta t = 10^{-14}$, $T = 500$ and in table 3 we can see that the energy is still conserved as desired.

3.4 Thermostat

Since the temperature drops from the initial value that we used to intialize the velocities, we want to implement a thermostat so we can change the velocities of the atoms in order for the system to reach a desired temperature. The so called Berendsen thermostat works by scaling the velocities of all atoms by a factor γ defined as

$$\gamma = \sqrt{1 + \frac{\Delta t}{\tau}\left(\frac{T_{bath}}{T} - 1\right)},$$

where τ is a parameter which adjusts how quickly the temperature should change. T is the temperature that we measure in the system and T_{bath} is the desired temperature. The thermostat will be usefull later when we want to create a phase diagram for argon. This scaling factor is implemented in the class *BerendsenThermostat*. We want to choose τ such that the time it takes to heat the system to the desired temperature is as fast as possible without overheating the system. With $\tau = \Delta t$ we get figure 5. We have to run the program for a while with the thermostat to make sure that the system is at the desired temperature. We also run the program without the thermostat for a while to ensure that the system is in its equilibrium.

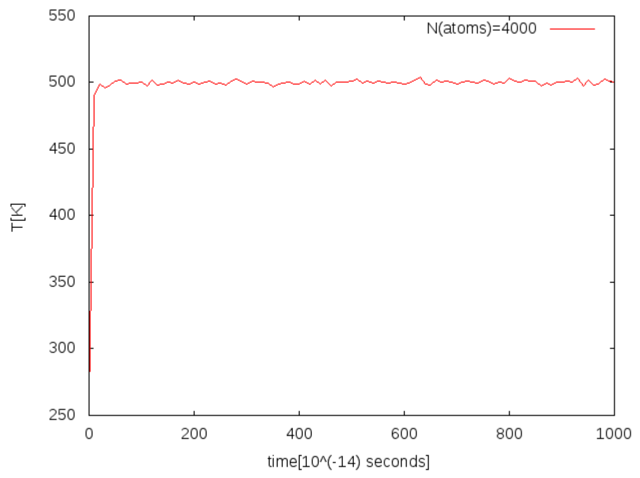


Figure 5: Temperature rise with thermostat

4 Phase transitions

We want to study the phase transitions of argon. Before we can use the program to generate a P-V diagram we have to measure the pressure of the system. We also measure the heat capacity before we finally present the P-V diagram for argon.

4.1 Measuring pressure and specific heat capacity

We now want to measure the pressure. In the NVT ensemble the pressure is given by

$$P = \rho_n k_B T + \frac{1}{3V}\left\langle \sum_{i=1}^{N} \mathbf{F}_i \cdot \mathbf{r}_i \right\rangle . \tag{16}$$

We want to express the pressure for pairwise interaction between the atoms. The sum can be written as

$$\sum_i \mathbf{r}_i \cdot \mathbf{F}_i = \sum_i \sum_{j \neq i} \mathbf{r}_i \cdot \mathbf{F}_{ij} = \frac{1}{2} \sum_i \sum_{j \neq i} (\mathbf{r}_i \cdot \mathbf{F}_{ij} + \mathbf{r}_j \cdot \mathbf{F}_{ji}) = \frac{1}{2} \sum_i \sum_{j \neq i} \mathbf{r}_{ij} \cdot \mathbf{F}_{ij} = \sum_{j>i} \mathbf{r}_{ij} \cdot \mathbf{F}_{ij} .$$

Using Newtons third law

$$\sum_{j>i} \mathbf{r}_{ij} \cdot \mathbf{F}_{ij} = -\sum_{i>j} \mathbf{r}_{ij} \cdot \mathbf{F}_{ij} .$$

Then we have the following for the pressure,

$$P = \rho_n k_B T - \frac{1}{3V}\left\langle \sum_{i>j} \mathbf{r}_{ij} \cdot \mathbf{F}_{ij} \right\rangle .$$

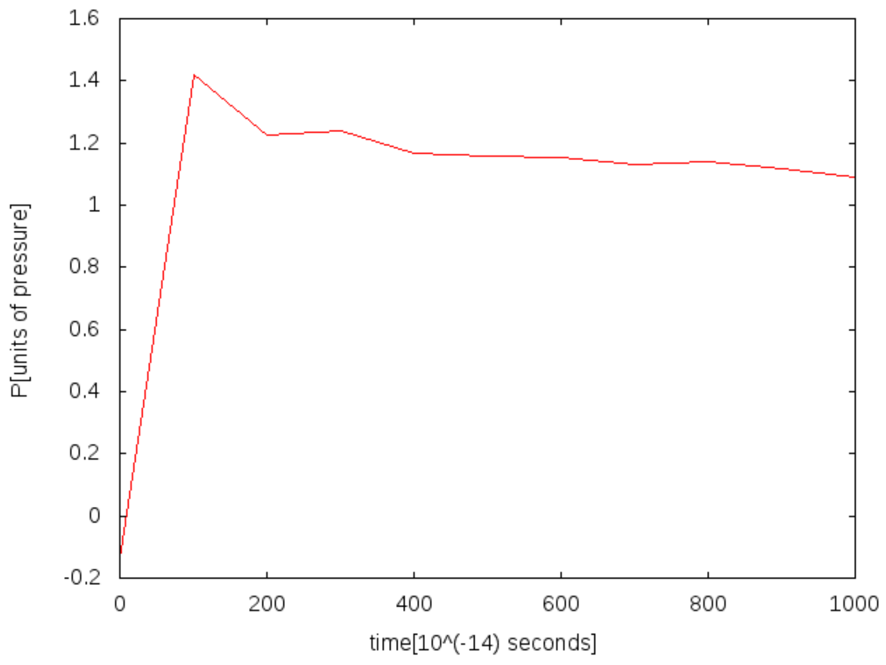


Figure 6: Pressure with $T_0 = 300$, $N_{atoms} = 4000$

As we can see in figure 6, the pressure initially rises and then goes down before the system reaches equilibrium. This is in agreement with what we saw initially for the energy and temperature earlier.

We want to estimate the heat capacity of argon in our model and compare it to experimental values. The specific heat capacity of argon at constant volume [3] is

$$c_v = \frac{312\text{J}}{\text{kgK}}.$$

We can estimate the heat capacity using

$$C_v = \frac{3k_B}{2}\left(\frac{1}{1 - \frac{2N_{atoms}}{3k_B T^2} V(E_k)}\right)$$

where $V(E_k) = \langle E_k^2 \rangle - \langle E_k \rangle^2$ is the variance of the kinetic energy. The specific heat capacity is defined as the heat capacity per unit mass

$$c_v \equiv \frac{C_v}{m},$$

where m in our case is the mass of one argon atom. Running the program with $N_{cells} = 15$, $b = 5.26$ and $T_{bath} = 500$ we get

$$c_v = \frac{312.187\text{J}}{\text{kgK}},$$

which correlates well with the experimental value.

[3]Tabulated values for heat capacities of different elements

4.2 P-V diagram

Finally we want to study the phase transitions of argon.

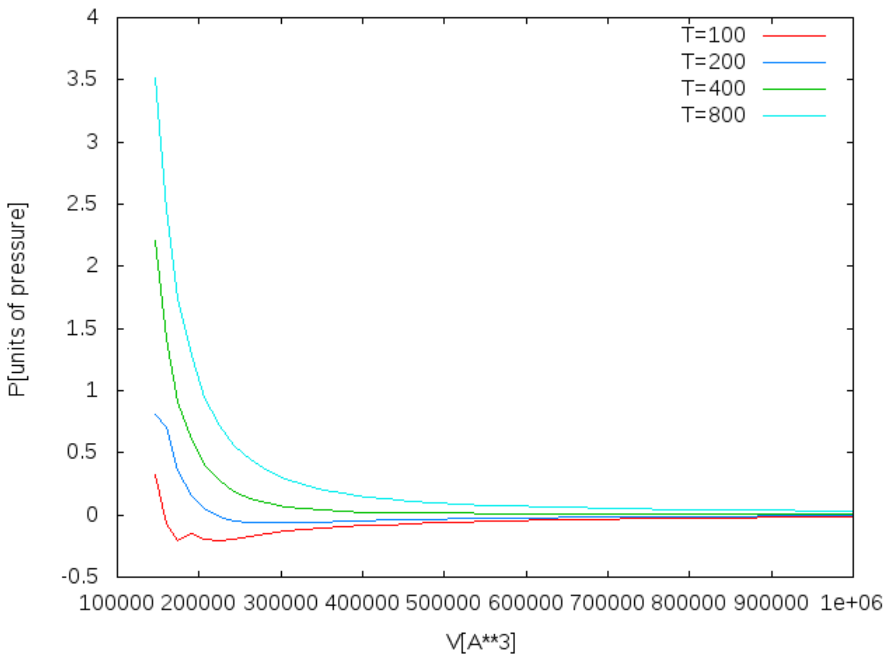


Figure 7: P-V Diagram, argon

In figure 7 we have generated a P-V diagram where we plot isotherms for different temperatures. An isotherm is a line in the P-V diagram where the temperature is kept constant while varying the volume of the system. The plot is generated by the script *Isotherms.py*.

In figure 8 we take a closer look at the isotherm for $T = 100$. We see that the pressure initially decreases when we increase the volume, but then the pressure increases again. This is actually unphysical. What actually happens is a phase transition. If we increase the volume slowly enough, quasistatic, the system would stay in equilibrium the whole time and it would actually be a straight line (constant pressure) through the entire transition. It resembles the diagram 9 one gets from the Van Der Waals model for phase transitions.

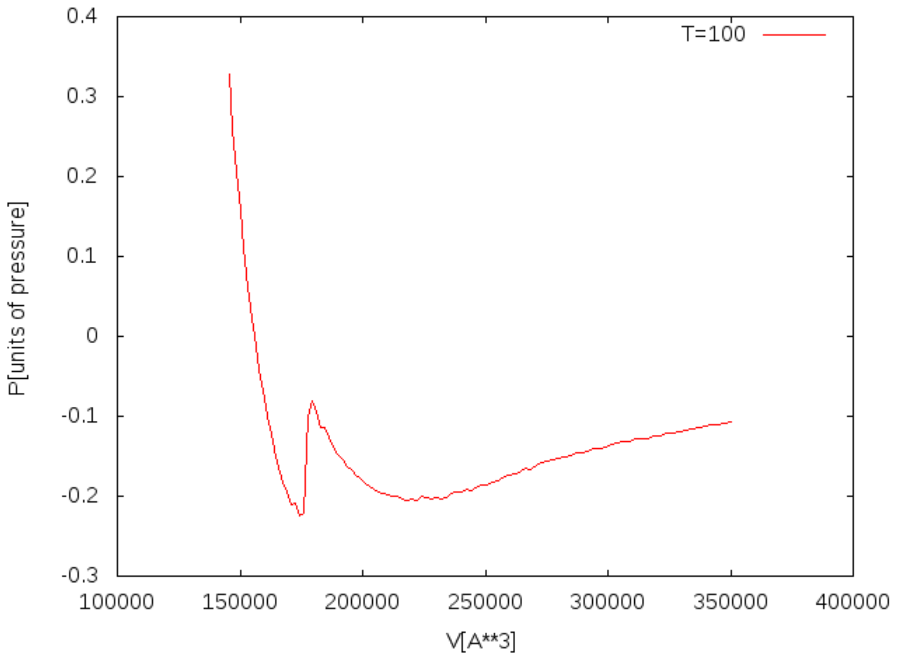


Figure 8: isotherm $T = 100$

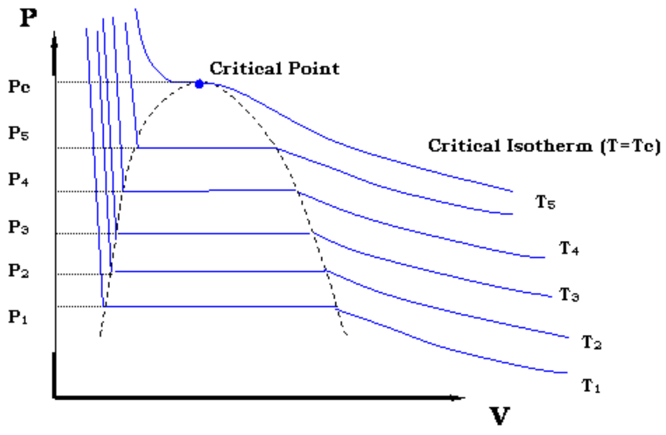


Figure 9: Van Der Waals Model

Hence, for $T = 100$ there is obviously a phase transition. For the other isotherms, where there are no unphysical behaviour, we simply have no transition. In conclusion, our model captures most of the same features as the Van Der Waals model.