Project 5 FYS4150

Vilde Mari Johansen and Kjetil Karlsen

December 6, 2017

Abstract

To do:
Write file every temp

Contents

1 Introduction

MD: "Numerical method for studying many-particle systems such as molecules, clusters, and even macroscopic systems such as gases, liquids and solids "

Kjetils notater: Simulere bevegelse N atomer og molekyler basert på klassisk newtonsk dynamikk: $F_i = m\frac{dri}{dt} = -\frac{\partial U}{\partial r:i}$. Kraften er ofte sterkt avstandsavhengig og kan komme fra eks. Lennard Jones potensial. [1].

Fysikk:

Material science: Can use potentials dervied from non-relativistic Schrödinger equation to investigate lattic and defect dynamics at atomistic scale. [2]

Kjemi: behavoiur ideal gas, chemical reacton $A + A \rightarrow B + B$ with hard sphere potential and Lennard Jones potential, [3]

Bio: Simulate membranes Molecular dynamics—the science of simulating the motions of a system of particles—applied to biological macromolecules gives the fluctuations in the relative positions of the

atoms in a protein or in DNA as a function of time. Knowledge of these motions provides insights into biological phenomena such as the role of flexibility in ligand binding and the rapid solvation of the electron transfer state in photosynthesis. Molecular dynamics is also being used to determine protein structures from NMR [4]

God link til "hva er MD": https://udel.edu/ arthij/MD.pdf

2 Theory

2.1 Crystal structure

There are different ways a solid could arrange itself in order to fill a given volume and minimize the overall energy of the structure. One of the densest is the face-centered cubic (FCC) structure of for instance NaCl. Assuming a hard sphere model of atoms, the FCC has together with the hexagonal close packed (HCP) the theoretically highest density of any structure. The unit cell of FCC is cubic with volume $V_{unitCell} = a^3$ made up by a basis of 4 atoms. These are in positions $a(0,0,0)$, $a(0,0,\frac{1}{2})$, $a(0,\frac{1}{2},0)$ and $a(\frac{1}{2},0,0)$. Due to symmetry, each of these atoms are repeated throughout the structure, as illustrated in figure 2.1. The corner atoms are repetitions of the $a(0,0,0)$ position, and each of these atoms are shared by 8 neighbouring cells. For the side atoms, they are shared only by two unit cells, giving at total of $6 \cdot \frac{1}{2} = 3$ side atoms per unit cell and $8 \cdot \frac{1}{8}$ corner atoms in the unit cell. The density of a unit cell size given in lengths of Angstrom, the density of a FCC cell is $\rho = \frac{4}{a^3} \text{Å}^{-3}$.

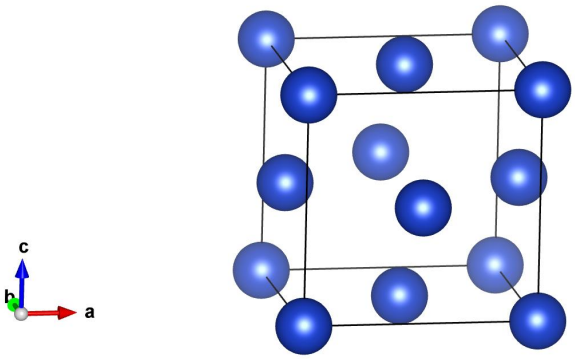

Figure 2.1: Illustration of a fcc structure. Although there are 14 atoms in this figure, the unit cell is made up of 4 atoms. The rest of these appear because of symmetry.

In numerical calculations it is not possible to calculate on a infinite lattice of atoms. As an infinite lattice have neighbouring atoms in every direction for every atom, an atom can not move into vacuum. By applying periodic boundary conditions (PBC's) to the position of the lattice, every atom moving out of the predefined supercell is put on the other side of the box. In one dimension, this is expressed as $r_{xi} = r'_{xi} - L$ if $r_{xi} > L$, with L being the supercell size. This ensures that the density is constrained and the volume of the supercell remain constant.

A surface experience different potential compared to bulk, as there are no neighbouring atoms in one of the directions, giving rise to different physics at the surface compared to the bulk. As infinite lattices does not have a surface, the surface effect needs to be minimized. By implementing the minimum image convention these effects can be reduced. This convention states that if a distance r_{ij} between two atoms i and j is larger than $\frac{L}{2}$, the distance between them is $r'_{ij} = r_{ij} - L$. Thus an

atom near an edge experiences forces of approximately equal magnitude to an atom near the centre of the supercell.

The PBC's and minimum image convention models an infinite supercell, by ignoring surface effects. However, it is not a true representation of an infinite cell. A perfect infinite cell consists of an infinite number of particles with random initial velocities, while a finite supercell will only have a finite number of atoms with sudorandom[1] initial velocities. These atoms interact, leading to a stronger codependence of the neighbours for the finite case. By increasing the supercell this effect will decrease.

2.2 Probability distribution

v-distribution: why Mbolztmann Discuss why the velocity of atoms are given from Maxwell-Boltzmann distrubution. - nonzero net momentum.

2.3 Lennard Jones potential

A very popular potential used in MD-simulations is the Lennard Jones potential, given in equation 1. It describes the potential energy as a function of distance between two objects i and j, r_{ij}.

$$U(r_{ij}) = 4\epsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right] \tag{1}$$

Through the relation $F(\mathbf{r} = -\nabla U(\mathbf{r})$, it is possible to determine the forces experienced by each atom by every other atom, simply as a function of the distance r_{ij} and relative position between two atoms. The distance r_{ij} is defined as $r_{ij} = |\mathbf{r}_{ij}| = |\mathbf{r}_i - \mathbf{r}_j|$. By investigating each spatial direction separately, a relation for the forces between two atoms i and j arises:

$$F_{ij}^{x} = -\frac{\partial U}{\partial r_{ij}}\frac{\partial r_{ij}}{\partial x_{ij}} \tag{2}$$

$$\frac{\partial r_{ij}}{\partial x_{ij}} = \frac{\mathrm{d}}{\mathrm{d}x_{ij}}|\mathbf{r}_i - \mathbf{r}_j| = \frac{\mathrm{d}}{\mathrm{d}x_{ij}}\sqrt{(\mathbf{r}_i - \mathbf{r}_j)^2} \tag{3}$$

$$= \frac{x_{ij}}{|\mathbf{r}_i - \mathbf{r}_j|} = \frac{x_{ij}}{r_{ij}} \tag{4}$$

$$\frac{\partial U}{\partial r_{ij}} = 4\epsilon\left[\frac{-12}{r_{ij}}\left(\frac{\sigma}{r_{ij}}\right)^{12} + \frac{6}{r_{ij}}\left(\frac{\sigma}{r_{ij}}\right)^{6}\right] \tag{5}$$

$$F_x(r_{ij}) = 24\epsilon\left[2\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right]\frac{x_{ij}}{r_{ij}^2} \tag{6}$$

By generalising equation 6 to a three dimensional space one gets a general expression of the force between two atoms i and j:

$$\mathbf{F}(r_{ij}) = 24\epsilon\left[2\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right]\frac{\mathbf{r}_{ij}}{r_{ij}^2} \tag{7}$$

[1] If a computer is used to generate the random initial velocities. For the theoretical infinite lattice one usually assumes true random initial conditions

Atom i experience a total force of $F(\mathbf{r}_i) = \sum_{j \neq i} \mathbf{F}(r_{ij})$. Each interaction is only necessary to compute once, as $\mathbf{r}_{ij} = -\mathbf{r}_{ji}$. However, it is necessary to update the force on each atom, according to Newtons third law. Equivalently, the total potential is given as $U_{tot} = \sum_i \sum_{j>i} U(\mathbf{r}_{ij})$.

Potential surface????

3 Method

3.1 Code structure

3.1.1 Overview of the classes

The code provided for the project were made up of several classes, each with different roles. To a large extent there is a hierarchy in the classes, with *main* being on the top. From this class all main processes are initiated, initial conditions are determined, and the integration loop is run. However, it is the class *atom* that facilitate the object in which to store information about each atom in the calculation. Under follows a quick overview of each class and the main purpose of it.

- *system*:
  Contains all the information about the system. Creates a system of atom-instances in a fcc lattice. It also contains the integration function, linking the system to *VelocityVerlet*. In addition, functions that are relevant to the behaviour of the system, like applying periodic BC's and calculation of forces are found here.

- *atom*:
  Contains information about each atom in the simulation, like starting position, position, velocity and force experienced by each atom. Is mainly a class to store information about each atom, but also contains a function to calculate the initial velocity distribution.

- *VelocityVerlet*:
  Contains simply the velocity verlet algorithm (per timestep). Takes the entire system class as an input variable and updates position, velocity and force on each atom in the system. Uses functions from *system* in order to update forces and utilize periodic BC's.

- *LennardJones*:
  When calling the *calculateForces* function in *system*, it links to this object. Updates forces on each atom in the system and the total potential, through the Lennard Jones potential.

- *statisticssampler*:
  Collect information about the system, like kinetic and potential energy, instantaneous temperature. In addition, also contains a unit test checking energy convergence.

- *vec3*:
  Defining the behaviour of every 3×1 vector utilized in the program. Also contains functions to calculate length of *vec3* objects, cross products, nullifying a *vec3* instance. Every vector in the code is a *vec3* instance.

- *unitconverter*:
  Contains information about constants and units used in the program, in addition to functions converting values between MD-units and SI units.

- *io*:
  Framework to create and store information to a *.xyz* file.

- *random.h*:
  Header file containing a selection of functions from the c++ library *random* and *ctime*.

3.1.2 Program flow

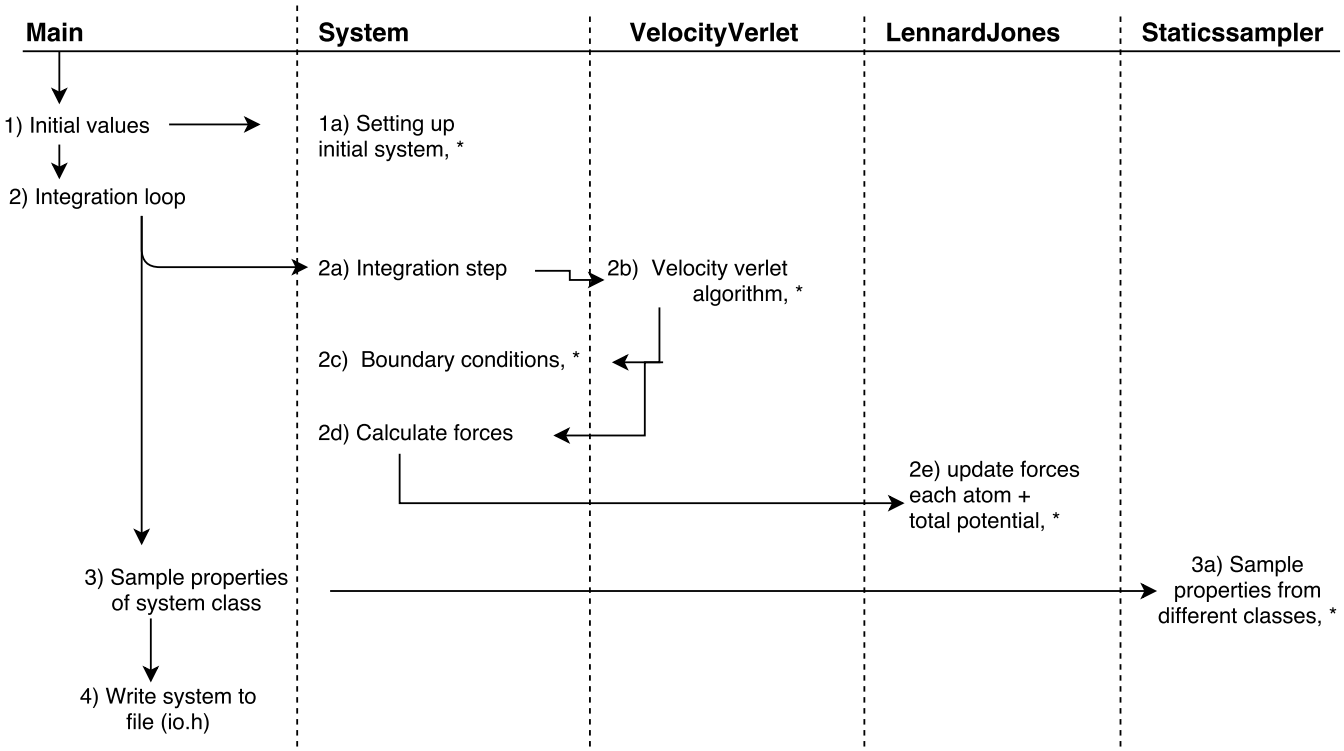

Figure 3.1: Simplified flow chart of the code structure. * indicates that the Atom class was used, see the discussion below

Figure 3.1 illustrates the work flow in the code somewhat simplified. Due to the amount of classes in this project, every time the *atom* class is called is marked by *. As *system* contains several atoms, each time *atom* is called is by a loop over all the atoms in *system*, updating i.e. the position. To some extent, all the classes can be viewed as a blackbox, with *main* controlling the sequence and what is being performed.

Step 1)specifies the initial values, like temperature and number of unit cells, sending them to step 1a, which creates the fcc lattice as specified by main. In addition, it also creates instances of *io* and *staticssampler* and initialize the output files. It also creates the specified amount of *atom*-instances. In order to integrate the system through time, a time loop is performed in *main*. However, each integration step is then micromanaged by *system* which has a defined instance of *VelocityVerlet*, which does the actual integration over every instance of *atom* in *system*. Through the instance "system" the periodic boundary conditions are applied and forces are calculated, see snippet below.

Listing 1: Snippet showing the main part of velocity verlet algorithm as applied in *VelocityVerlet*

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

```
9
10 for(Atom *atom : system.atoms()) {
11  atom->velocity += dthalf*atom->force/(atom->mass());
12  }
```

The sampling of properties are done in step 3, by applying the "sample" function in *Staticssampler*, 3a. This function updates the private variables of *Staticssampler*, describing properties like temperature and kinetic energy. This is done by calling several specific *Staticssampler* - functions sampling each property individually. The last step, step 4, writes .xyz-file through the *io* instance "movie" in *main*. In addition, the information stored in *Staticssampler* is saved in a separate output file. This should be done in *io* for the most logical flow, but we found it simplest to do this through *Staticssampler*.

3.2 Bugs and unit tests

As the structure of the program was provided, a large array of unit tests were not necessary. However, two simple test became very useful. The first, checking that the total momentum was set to 0 provided useful input, as the initial version of "removeTotalMomentum" did not take the changing unit cell size into account. Secondly, the unit test for the conservation of energy revealed several mistakes. One example of this is that we counted the potential energy per atom in interacting pair, instead of each pair. A more serious bug were found in *velocityVerlet* through this unit test. Line 5 in snippet 1 shows the original, first position update in the algorithm. Here the velocity of each atom is divided by the atom mass, which has no place in the relationship $\frac{dv}{dt} = x(t)$. Most likely this is due to a copy and paste error of the line above, simply substituting the force with velocity.

4 Result

5 Discussion

6 Conclusion

References

[1] Vincent E. Lamberti, Lloyd D. Fosdick, Elizabeth R. Jessup, and Carolyn J. C. Schauble. A hands-on introduction to molecular dynamics. *Journal of Chemical Education*, 79(5):601, 2002.

[2] Martin O Steinhauser and Stefan Hiermaier. A review of computational methods in materials science: Examples from shock-wave and polymer physics. *International Journal of Molecular Sciences*, 10(12):5135–5216, 12 2009.

[3] J. Gorecki and J. Gryko. Molecular dynamics simulation of a chemical reaction. *Computer Physics Communications*, 54(2):245 – 249, 1989.

[4] Martin Karplus and Gregory A. Petsko. Molecular dynamics simulations in biology. *Nature*, 347:631 EP –, 10 1990.

Appendix