Fys4460 – 2013 – Project 1

Introductory Molecular Dynamics Modeling

Through this project we will implement a molecular dynamics code to model the behavior of a system of Argon atoms, and use this model to study statistical properties of the system.

Molecular dynamics – model development

We will start by developing programs to generate Argon atom systems, visualize the systems, find the motion of the atoms for a simple interatomic potential, and measure and characterize average properties of the system.

State of Molecular Dynamics system

In the models we will be using here, the state of an atomic system is given by the positions, velocities, and types of the atoms involved. There are no internal states in the atoms or in their interactions. This means that we only need to prescribe the positions and velocities of the atoms at each time to provide a complete description of the development of the system. Our simulator will consist of several parts: some parts generate, modify or measure the state in various ways, and one part propagates the state forward in time – the molecular dynamics simulator. It is therefore important for you to develop a good way to store the state, so that it is compact if you want to store many states for a large system, and so that we easily can visualize the state.

State files

You can store the state in binary or ascii format. In this project, it is sufficient to store it in ascii, but later we will need to introduce a binary storage format. If we choose an ascii format, we can choose a format that can easily be visualized.

A good choise for an ascii file format is the simple atom position file format that can be used with VMD, the `.xyz` format. It contains only the chemical element of all atoms, their cartesian positions and optionally their velocities. Usually, the positions are given in units of angstroms (Å). The first line is the number of atoms in the system, and the second line is a comment. Example:

```
8
Argon atoms at the corners of a unit cube, with zero velocity!
Ar 0 0 0 0 0 0
Ar 1 0 0 0 0 0
Ar 0 1 0 0 0 0
Ar 0 0 1 0 0 0
Ar 1 1 0 0 0 0
Ar 1 0 1 0 0 0
Ar 0 1 1 0 0 0
Ar 1 1 1 0 0 0
```

You can save this file and load it into VMD, using

File→New molecule

and

Graphics→Representations

to change the representation to a useful format.

VMD allows the visualization of dynamics by including several time steps, either by writing the data (including the header) can be written subsequently in the same file, or by a sequence of numbered files with extensions `.001`, `.002` etc. with one file for each time step.

In general it is useful to store each state in a directory and write scripts that extract a sequence of files from each directory into a common directory for visualization or processing.

Crystal structure

We will start the simulation with all the atoms on a face-centered cubic lattice, since this corresponds to the crystalline structure of Argon. A lattice is described by a set of bases vectors, $\hat{u}_n$, $n = 1, 2, 3$ and each cell is given by a linear sum with integer prefactors for each base vector. The positions of the cells in a $N_x \times N_y \times N_z$ lattice are therefore

$$\vec{R}_{i,j,k} = i\hat{u}_1 + j\hat{u}_2 + k\hat{u}_3 \ . \tag{1}$$

where $i = 1, 2, \ldots, N_x$, $j = 1, 2, \ldots, N_y$, $k = 1, 2, \ldots, N_z$.

For a face-centered cubic lattice the base vectors are the cartesian unit vectors multiplied by the length b of the unit cell:

$$\hat{u}_1 = b\hat{i} \ , \ \hat{u}_2 = b\hat{j} \ , \ \hat{u}_3 = b\hat{k} \ , \tag{2}$$

The positions of the atoms within a cell are given relative the base cell position. For a face-centered cubic lattice the positions of the atoms relative to the origin of the cell are:

$$\vec{r} = 0\,\hat{i} + 0\,\hat{j} + 0\,\hat{k} \ , \tag{3}$$

$$\vec{r} = \frac{b}{2}\hat{i} + \frac{b}{2}\hat{j} + 0\,\hat{k} \ , \tag{4}$$

$$\vec{r} = 0\,\hat{i} + \frac{b}{2}\hat{j} + \frac{b}{2}\hat{k} \ , \tag{5}$$

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

(a) Write a program that generates an $N_c \times N_c \times N_c$ unit cell face centered cubic lattice of Argon atoms, saves it to a state file. If you do not use the `.xyz` format for the state files, you also need to write a program to convert the state file to a `.xyz` file for visualization. Visualize the lattice for $N_c = 8$. For solid Argon the lattice constant is $b = 5.260$Å.

The state consists of both positions and velocities for all atoms. We must therefore modify the program that generates the initial state to also generate velocities for all the atoms. From statistical physics, we know that in equilibrium at temperature T, the velocities of the atoms are given by the Boltzmann distribution, which means that the x-, y-, and z-components of the velocity are normally distributed with average zero and standard deviation $\sqrt{k_\mathrm{B}T/m}$, where m is the mass of an atom and k_B is the Boltzmann constant. For Argon $m = 39.948$amu, and you may start the system at $T = 100$K.

(b) Modify the program to also generate initial velocities of all the atoms. Remove any initial linear momentum from the system. Write a program that reads the state file and plots the distribution of velocities in the x, y, and z directions, as well as the distribution of the magnitude of the velocity. Check that the velocity distribution in the generated state has the right average and standard deviation.

Motion

Integrator

We use the symplectic and numerically stable velocity Verlet algorithm to integrate particle motion. Even though this integrator is simple, it is time reversible and provides excellent energy conservation – you generally cannot do much better for your production codes.

For each particle i, the steps are as follows (currently setting $U_i = 0$):

$$\mathbf{v}_i(t+\Delta t/2) = \mathbf{v}_i(t) + \frac{\mathbf{F}_i(t)}{2m}\Delta t \tag{7}$$

$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + \mathbf{v}_i(t+\Delta t/2)\Delta t \tag{8}$$

$$\mathbf{F}_i(t+\Delta t) = -\nabla_i U_i(\{\mathbf{r}\}(t+\Delta t)) \tag{9}$$

$$\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t+\Delta t/2) + \frac{\mathbf{F}_i(t+\Delta t)}{2m}\Delta t \tag{10}$$

See *Computational Physics* by J.M. Thijssen for more details.

You should write your integrator so that it acts on a given state and produces a set of new states so that you may start the simulation again from any of the states generated.

(c) Integrate the dynamical equation (N2L) using this method. Visualize the dynamics, and describe what happens.

Periodic boundary conditions

The particles will now spread out into space. We are only interested in bulk atoms in a material, so the next step is implementing periodic boundary conditions.

(d) Rewrite your program to include periodic boundary conditions. Every time the position of a particle is updated, the program must check if it has gone though one of the sides, and rescale the particle position correspondingly.

Forces

We use a Lennard-Jones potential for the interatomic interactions. The Lennard-Jones potential has the following form:

$$U(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{11}$$

where r is the distance between the atoms. For Argon optimal values of the parameters are:

$$\frac{\epsilon}{k_\mathrm{B}} = 119.8\mathrm{K}\ ,\ \sigma = 3.405\text{Å}\ . \tag{12}$$

(e) Find the Lennard-Jones force on an atom, and the equilibrium interatomic distance.

Internally in our integrator we use MD units. These assume that all the particles in a simulation are identical, so the masses and Lennard-Jones parameters can be factored out of the equations. Here, we use σ as the unit for length and ϵ as units for energy.

(f) Rewrite the equations of motion using MD units. Check that the force now only includes numerical values.

(g) Implement the Lennard-Jones force in your modeling program in order to model the behavior of Argon. Use the minimum image convention to handle periodic boundary conditions when calculating distances. Ensure that states are stored in a convenient format.

In the *minimum image convention*, the distance between atoms/atom replicas i and j in the x direction becomes $\min_\delta (x_i - x_j + \delta L)$ where $\delta \in \{-1, 0, 1\}$ and L is the length of the simulation box in the x direction. This limits the interaction range to half the system size, which is more than enough for our potential.

You should now have a working MD program for simulating bulk Argon in its solid, liquid and gas phases.

Neighbour lists

The force calculations are the most time consuming part of your program. The number of force terms is $\frac{1}{2}N(N-1)$ for each time step, which gives a workload scaling $\propto N^2$. We want to improve this by neglecting force terms for particles that are far apart. The L-J interaction is short ranged and can be neglected for distances over $r_{\text{cut}} \approx 3\epsilon$. A simple and efficient way of achieving this is by implementing *Verlet lists*, but this is not a good method when we want to develop a parallel implementation of the simulator. Instead, we will use cell lists. We divide space into cells of size r_{cut}. In each cell we keep a (linked) list of all the atoms in that cell. An atom will only interact with atoms in the 26 neighboring cells. To find the interactions for an atom we need to find which cell it belongs to, and then calculate the interactions with all atoms in 27 cells, the 26 neighboring cells and the cell the atom is in. This reduces calculation time significantly, and it also makes parallellization simple.

(h) Implement the cell list method in your program, and compare how long time the program uses to simulate a given number of time steps for various system sizes.

Macroscopic observables

We can use the MD simulation to measure macroscopic quantities by averaging properties of the simulation system. The ergodic hypothesis states that the time a system has one particular value of an observable A is proportional to the phase space volume where A has this value. This applies to systems in equilibrium studied for a long period of time. As a result, the time average and ensemble average of a variable are equal. If we average over long enough periods of time, and our microscopic interaction model is correct, we can use the models to predict equilibrium properties of real systems. In addition, we can use the simulation to gain insight into the statistical properties of systems with many particles, and in particular into the fluctuations in such systems.

(i) According to the central limit theorem, the velocity distribution of the particles will eventually evolve into a Maxwell-Boltzmann distribution independent of the initial conditions. Let us test this by starting the simulation with velocities that are uniformly distributed random numbers in the interval $[-v, v]$, for a reasonable v. Investigate the probability density for the velocities and for the speeds of the atoms at a given time by writing the velocities to a file and using the Matlab `hist()` function. You can study the time-development of the velocity distribution to estimate how long time it takes for the velocities to reach a Maxwell-Boltzmann distribution?

We are modelling a micro-canonical ensamble, since we study systems with constant volume and a constant number of particles, and, in theory, constant energy. In practice, though, our integration scheme does not conserve energy exactly.

(j) Find the total internal energy (kinetic and potential) of the system, $E(t)$, and plot it as a function of time. How does the size of the fluctuations in energy depend on the time-step Δt?

In general, it is not trivial to calculate the temperature for general potential forms. The simplest estimate assumes equilibrium between the translational and potential degrees of freedom. According to the equipartition principle, the average total kinetic energy is

$$\langle E_k \rangle = \frac{3}{2} N k_{\mathrm{B}} T \tag{13}$$

where N is the number of atoms and T is our estimate for the system temperature.

(k) Use this method to measure the temperature as a function of time, $T(t)$. (Don't forget to equilibriate the system first). Find the average temperature of the system (after equilibration), and compare with the temperature you used to generate the initial velocity distribution. Characterize the fluctuations in temperature, and find how the fluctuations vary with system size.

There are several ways of measuring the pressure P of a many-atom system. The method we will use is derived from the virial equation for the pressure. In a volume V with particle density $\rho = N/V$, the average pressure is

$$P = \rho k_{\mathrm{B}} T + \frac{1}{3V} \sum_{i<j} \vec{F}_{ij} \cdot \vec{r}_{ij} \tag{14}$$

where the sum runs over all interacting particle pairs. Note that this expression depends on the ensemble – and is valid for the micro-canonical ensemble only. The vector products should be computed and summed up inside the force loops for efficiency.

(l) Measure the pressure as a function of temperature in your system, plot $P(T)$, and discuss the result. How do they compare with your theoretical expectations?

(m) (Optional) Measure the pressure as a function of both temperature and density, visualize and discuss the results.

The diffusion constant

We want to characterize transport in a fluid by measuring the self-diffusion of an atom: We give an atom i a label, and measure its position as a function of time, $\vec{r}_i(t)$. We find the diffusion constant from the mean square displacement of all atoms (we trace the motion of every atom):

$$\langle r^2(t) \rangle = \frac{1}{N} \sum_{i=1}^{N} (\vec{r}(t) - \vec{r}_{\mathrm{initial}}). \tag{15}$$

From theoretical considerations of the diffusion process we can relate the diffusion constant in the liquid to the mean squear displacement through:

$$\langle r^2(t) \rangle = 6Dt \text{ when } t \to \infty \ . \tag{16}$$

This result is similar to the behavior of a random walker in three dimensions, which is indeed a good approximation to the motion of an atom in a fluid.

(n) Plot the mean square displacement as a function of time and estimate the diffusion constant, D. Investigate the effect of temperature by finding $D(t)$ for some temperatures of your own choice in the liquid phase of argon. Remember that you are measuring the total distance travelled by the atoms, which must be continous when an atom is displaced though the periodic boundaries!

Microscopic structure – radial distribution functions

The radial distribution function $g(r)$, also called a pair correlation function, is a tool for characterizing the microscopic structure of a fluid. It is interpreted as the radial probability for finding another atom a distance r from an arbitrary atom, or equivalently, the atomic density in a spherical shell of radius r around an atom. It is commonly normalized by dividing it with the average particle density so that $\lim_{r\to\infty} g(r) = 1$.

(o) Estimate $g(r)$ for $r \in (0, \frac{L}{2}]$ in your Argon system. The easiest way is to divide the distance interval into bins, loop over all pairs of particles and count how many distances belong in each bin. Time-averaging the function gives a better description of the system's general behaviour. Plot $g(r)$ for temperatures where the system is in solid and liquid phases. Does it appear as expected? How would the exact $g(r)$ look for a perfect crystal?

Thermostats

In order to simulate the canonical ensemble, interactions with an external heat bath must be taken into account. Many methods have been suggested in order to achieve this, all with their pros and cons. Requirements for a good thermostat are:

- Keeping the system temperature around the heat bath temperature
- Sampling the phase space corresponding to the canonical ensemble
- Tunability
- Preservation of dynamics

The method closest to fullfilling these requirements which is in widespread use is the Nos-Hoover thermostat, which is somewhat complicated to implement. We will focus on simpler methods. They will require negligble CPU time and should be applied for each time step.

Many thermostats work by rescaling the velocities of all atoms by multiplying them with a factor γ. The Berendsen thermostat uses

$$\gamma = \sqrt{1 + \frac{\Delta t}{\tau}\left(\frac{T_{\text{bath}}}{T} - 1\right)} \tag{17}$$

with τ as the relaxation time, tuning the coupling to the heat bath. Though it satisfies Fourier's law of heat transfer (the transfered heat between two bodies is proportional to their temperature difference) it does a poor job at sampling the canonical ensemble.

(p) Implement the Berendsen thermostat as a function in your code. $\tau = \Delta t$ will keep the (estimated) temperature exactly constant. It should be put to 10-20 times this value. Describe its impact on the observed motion of the atoms.

The Andersen thermostat simulates (hard) collisions between atoms inside the system and in the heat bath. Atoms which collide will gain a new normally distributed velocity with standard deviation $\sqrt{k_B T_{\text{bath}}/m}$. For all atoms, a random uniformly distributed number in the interval $[0, 1]$ is generated. If this number is less than $\frac{\Delta t}{\tau}$, the atom is assigned a new velocity. In this case, τ is treated as a collision time, and should have about the same value as the τ in the Berendsen thermostat. The Andersen thermostat is very useful when equilibrating systems, but disturbs the dynamics of e.g. lattice vibrations.

(q) Implement the Andersen thermostat, and compare $T(t)$ graphs for simulations using the two methods. Again, be aware that our T is just an approximation to the real temperature. Describe the impact of the thermostat on the observed dynamics of the system of atoms – as seen in your visualizations of the dynamics.

Appendix

Units

In all calculations, we will use so-called MD units. These assume that all the particles in a simulation are identical, so the masses and LJ parameters can be factored out of the equations. You will need to insert $A = \bar{A}A_0$ for every variable quantity A in equations 25-28 and 29 above. For example, for velocity, $v = \bar{v}\frac{L_0}{t_0}$. The time step Δt must also be treated this way. All non-numerical constants should disappear, except for in the velocity distribution standard deviation.

Quantity	Conversion factor	Value
Length	$L_0 = \sigma$	3.405
Time	$t_0 = \sigma\sqrt{m/\epsilon}$	$2.1569 \cdot 10^3$ fs
Force	$F_0 = m\sigma/t_0^2 = \epsilon/\sigma$	$3.0303 \cdot 10^{-1}$ eV/
Energy	$E_0 = \epsilon$	$1.0318 \cdot 10^{-2}$ eV
Temperature	$T_0 = \epsilon/k_\mathrm{B}$	119.74 K

Table 0.1: Conversion factors A_0 from MD units for variable quantities.

In case you want to convert between your internal MD units and other units during input and output, the actual values of the conversion factors are listed in table 0.2. These are calculated using the argon mass, lattice constant and LJ parameters: $m = 39.948$ amu, $a = 5.260$ (solid argon), $\sigma = 3.405$, $\epsilon = 1.0318 \cdot 10^{-2}$ eV. Another common practice is putting $E_0 = 4\epsilon$, affecting the conversion factors F_0, T_0 and t_0.

Normally distributed numbers

Normally distributed random numbers are obtained by performing a Box-Muller transform on uniformly distributed numbers. Let u and v be uniform numbers in the interval $(-1, 1)$. These numbers will only be accepted for the transformation if $s = u^2 + v^2$ is in the interval $(0, 1)$. In that case, we obtain two normally distributed numbers n_1 and n_2 by multiplying u and v with a constant,

$$n_1 = Su, \qquad n_2 = Sv, \qquad S = \sqrt{\frac{-\ln s}{s}}. \tag{18}$$

n_1 and n_2 will have standard deviations of 1, but multiplying all generated numbers with a constant will give a distribution with that constant as the standard deviation.

VMD output files

A simple atom position file format that can be used with VMD is the `.xyz` format. It contains only the chemical element of all atoms, their cartesian positions and optionally their velocities. The first line is the number of atoms in the system, and the second line is a comment. Example:

```
8
Argon atoms at the corners of a unit cube, with zero velocity!
Ar 0 0 0 0 0 0
Ar 1 0 0 0 0 0
Ar 0 1 0 0 0 0
Ar 0 0 1 0 0 0
Ar 1 1 0 0 0 0
Ar 1 0 1 0 0 0
Ar 0 1 1 0 0 0
Ar 1 1 1 0 0 0
```

When visualizing several time steps, the data (including the header) can be written subsequently in the same file. Alternatively, one file can be used for each time step, with extensions `.001`, `.002` and so on.

The most relevant dialogues in the VMD interface are File→New molecule and Graphics→Representations.

ParaView output files

ParaView reads the legacy `.vtk` format, which can contain geometry information and low-rank tensor values for almost any kind of system. The complete description of this format can be found at `www.vtk.org/VTK/img/file-formats.pdf`. As it will require time to create such an output routine, a class for outputing point particle data to `.vtk` files is provided. You can choose to use this or modify it to fit your own program better.

For each output file, an output object of the subclass `vtk_points` must be created. The first constructor argument is the output file name. For visualization in ParaView, the files must be named with a time step number at the end, e.g. `md21.vtk` or `md0021.vtk`. The second argument is a comment line, and the third one a boolean determining whether binary output should be used instead of ASCII. ASCII files are human-readable and can be used for testing, but binary files are smaller and faster read by ParaView.

Next, the geometry (atom positions) must be specified. The method `geometry` has two parameters, the array of positions (indexed by atom index and direction) and the number of atoms. Scalars and vectors for each atom can be specified using the `writescalars` and `writevectors` methods, respectively. The arguments are the actual scalar or vector arrays and a variable name describing the data. Example of usage:

```
ostringstream namemaker;
namemaker << "md" << timestep << ".vtk";
vtk_points mywriter(namemaker.str().c_str(), "My MD data!", true);
mywriter.geometry(current_positions, num_atoms);
mywriter.writevectors(current_velocity, "Velocity");
mywriter.writescalars(potential_energy, "Pot_energy");
```

Glyph and Histogram are the filters in ParaView that are most relevant to us.

End of Project 1