Molecular Dynamics: Lennard-Jones liquid

Florian Dommert, Nadezhda Gribova, Peter Košovan*

15th November 2011

1 Introduction

In the previous tutorial we were simulating the solar system and were examining the trajectories of planets. In this tutorial, we will simulate a much denser system of soft spherical particles. Contrary to the planetary system, this one is dominated by collisions and therefore individual particle trajectories strongly depend on initial conditions. This is typical for molecular systems, where the statistical properties of the whole system rather than the the exact trajectories are studied.

The focus of the tutorial is to follow how the system equilibrates. In the first example, we will show that a system such as a dilute gas or a liquid reaches within several simulation steps a state with the same statistical properties, irrespective of the initial configuration. In the second example we show that in some systems such as a crystal or a glass this may not happen within an acceptable simulation time.

2 Implementation of the Force Field

We are going to simulate a system consisting of particles interacting with the Lennard-Jones (LJ) potential. This potential has been originally derived for a system of noble gas atoms, where the interactions between induced dipoles in the atoms dominate on longer distances. It can be shown that this interaction has an r^{-6} dependence, which is the second term in Equation 1. About the short-ranged repulsive contribution it is only known that it is much steeper than the attractive part. Hence, for computational convenience, r^{-12} has been used for the repulsive part, leading to the final potential:

$$V_{\mathrm{LJ}}(r) = 4\epsilon \left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right) . \tag{1}$$

If we wanted to simulate the full LJ potential, we would quickly run into problems, since its interaction range is infinite and we can only simulate a finite system. It converges

*kosovan@icp.uni-stuttgart.de

fast enough to zero so that we can introduce an approximation by truncating it at a certain distance and adding an appropriate shift so that it is continuous:

$$V_{\mathrm{LJ}}(r) = \begin{cases} V_{\mathrm{LJ}}(r) - V_{\mathrm{LJ}}(r_{\mathrm{cutoff}}) & r \leq r_{\mathrm{cutoff}} \\ 0 & r > r_{\mathrm{cutoff}} \end{cases} . \tag{2}$$

For $r < r_{\mathrm{cutoff}}$ the forces due to the truncated LJ potential are the same as due to the full form. In practice, $r_{\mathrm{cutoff}} = 2.5\,\sigma$ is often used. To obtain thermodynamic properties of a system interacting with the full LJ potential from a simulation performed using the truncated version, a correction has to be added. If the cutoff radius r_{cutoff} is chosen large enough, a correction to the total energy can be estimated:

$$E_{\mathrm{corr}} = \int_0^{r_{\mathrm{cutoff}}} V_{\mathrm{LJ}}(r_{\mathrm{cutoff}}) r^2 g(r) \mathrm{d}r + \int_{r_{\mathrm{cutoff}}}^{\infty} V_{\mathrm{LJ}}(r) r^2 g(r) \mathrm{d}r \,, \tag{3}$$

where $g(r)$ (see also further below) obtained from the simulation is used. Because $g(r \to \infty) = 1$, the integral converges only for potentials decaying faster than r^{-3}. For interaction potentials decaying slower than r^{-3}, special methods have to be used.

2.1 Reduced units

For simple liquids composed of point-like particles which interact with the same pairwise-additive potential of a general form:

$$U(r) = \epsilon \phi(\sigma / r) \,, \tag{4}$$

where ϵ and σ are constants and ϕ is a smooth and differentiable funcion, it is possible to define a set of dimensionless reduced units such as $x^* = x/\sigma$, $V^* = V/\sigma^3$, $T^* = k_{\mathrm{B}}T/\epsilon$ and $P^* = P\sigma^3/\epsilon$. Within this set of reduced units, all systems interacting with potentials of the form given by equation (4) follow one single universal equation of state. This law is called the Theorem of corresponding States. Therefore, results of simulations of simple fluids are presented in reduced units. One particular implication of the theorem of corresponding states is that we can perform all simulations with $\epsilon = 1.0$ and $\sigma = 1.0$ and if desired, the results can be transfered to other values of interaction parameters. For computational precision, it is desirable to choose the reduced units in the natural units of the problem under consideration, which makes all measured quantitites on the order of unity.

Task: (2 points)

- *Modify the file* `force.c` *to enable the calculation of the interaction energies and forces due to the truncated LJ interaction potential with with* $\epsilon = 1.0$, $\sigma = 1.0$ *and* $r_{cutoff} = 2.5$.
- *In the implementation, first the value of* $|\vec{r}||\vec{F}|$, *is computed and stored in variable* `Ff`, *then it is divided by* r^2 *and later on multiplied by the components of* $\vec{r}$: (x, y, z). *This is definitely not the most straightforward way. Try to explain why one should prefer this implementation rather than directly computing* $|\vec{F}|$ *and multiplying it by components of a unit vector in the direction of* $\vec{r}$: $(x/r,\ y/r,\ z/r)$.

Useful things to note:

- Pay attention: so far we have provided the potentials but not forces!
- All necessary variables have been declared for you, only insertion of some lines is required.
- When the implementation is done, type `make` to produce the executable `md_nve`.

3 Running the simulation

3.1 Input of parameters

The input parameters for the simulation program are provided via the input file `input.in` This is one of the usual ways of providing input to simulation programs. This file requires the input parameters in a fixed order and only little checking of the sanity of the input parameters is performed.

The time evolution of observables is written into file `observables.dat` Furthermore, files containing the measured $g(r)$ (`rdf.dat`), velocity distribution (`VelDist.dat`), simulation trajectory (`traj.vtf`), velocity autocorrelation function (`vacf.dat`) and mean-square displacement (`msd.dat`) are written out. If files with the same names are already present, they are overwritten. The last two output files are produced for historical reasons and will not be discussed in the tutorial. When performing a series of simulations it is a good practice to rename the output files to names which contain the parameters at which the simulation has been performed. To visualize the simulation trajecotry, execute

```
vmd -e vmd_movie.script
```

If you have implemented the forces and your code compiles without an error, run a simulation and plot the total energy from the output file `observables.dat`. If the total energy is not conserved, search for a mistake in your code. If you do not switch off the velocity rescaling by setting the number of rescaling steps to zero, energy will not be conserved while rescaling is performed (see also further below). On the other hand, if

you switch off the velocity rescaling, you may need to switch on the pre-equilibration to prevent the system from blowing up due to very unfavourable initial configuration.

3.2 Initial system setup

The first step when starting a simulation is to set up the system. In the code provided to you, there are three options of initial system configuration:

1. completely random setup
2. simple cubic lattice
3. hexagonal close-packed lattice

If you have done everything right and the simulated system is well behaved, the ensemble averages from the simulation should not depend on the initial setup. We will test this in one of the following tasks.

3.3 Equilibration

Imagine you want to measure the density of water at ambient conditions and you have two samples: one stored in a freezer and one on a shelf. If you take the one from the freezer and do the measurement immediately, the result will differ from the one on the shelf. If you leave the frozen sample on a table for some time and let it come to equilibrium with the surroundings, the result will be the same as for the one from the shelf. In a simulation we are in a similar situation. Usually we do not know at the beginning what the system looks like under the simulation conditions. On the contrary, we are often performing the simulation to answer this question. We prepare the system in an arbitrary initial configuration and let it equilibrate with a virtual heat bath.

Equilibration is manifested by a time-drift of all observables. On the contrary, once the equilibrium state is reached, *all* observables fluctuate around a constant value. Some observables relax to equilibrium quickly while others may take much longer. How long the equilibration takes depends on system properties as well as on how far from equilibrium it is. A system cannot be considered as equilibrated if there exists one single observable which exhibits a time-drift.

Unfortunately, there is no universal recipe, according to which one can safely tell that the system has equilibrated. When the measured quantities do not drift anymore, most probably the system is equilibrated but this is no guarantee! To decide if we can start collecting data, we have to make sure that the time over which the observables are fluctuating around a constant value exceeds many times the estimated duration of equilibration. We may also need to apply some physical knowledge of the simulated system.

When we know that the starting configuration is very far from equilibrium, it may be useful to use some other method to bring it quickly closer to equlibrium before the simulation is started. One such method, which is implemented in our code, is called

steepest descent energy minimization. It is switched on and off by setting the appropriate variable in the input file to 1 or 0.

3.4 Data collection and measurement of observables

After the system has equilibrated, we can start measuring observables which will then be used for the computation of ensemble averages. To measure observables in MD, we make the use of the ergodic hypothesis which states that the ensemble-average of an observable is equal to its time-average over a long-enough time interval:

$$\langle A \rangle = \lim_{t_{\max} \to \infty} \frac{1}{t_{\max}} \int_0^{t_{\max}} A(t)\, \mathrm{d}t \tag{5}$$

In other words, if we follow the evolution of the system long enough, it visits each point of the configurational space with the proper probability corresponding to the simulated statistical ensemble. If we measure instantaneous values of A at regular intervals Δt then the ensemble average is computed from simulation as

$$\langle A \rangle = \frac{1}{N} \sum_{i=0}^{N} A(i\Delta t)\,. \tag{6}$$

When computing ensemble averages, data from the equilibration have to be discarded. In the current code, we measure the observables on the fly while the simulation is running and compute the ensemble average at the end. The variable `Nsteps_equil` tells the program after how many time steps it starts collecting data to compute the averages. When starting the simulation, we do not know how long the equilibration will take but we can make an educated guess, e. g.from a short test run. It is always necessary to check at the end, that the equilibration has been shorter than `Nsteps_equil` we set at the beginning and if not, we have to rerun the simulation again with a higher value of `Nsteps_equil`.

In the following we will introduce formulae which are derived from statistical mechanics to obtain some observables from the simlation. To measure the temperature, we make the use of equipartition theorem stating that kinetic energy per degree of freedom is $1/2\, k_{\mathrm{B}}T$. Point-like particles have 3 translational degrees of freedom each, and hence the temperature is computed as follows:

$$T = \frac{2}{3} \frac{U_{\mathrm{kinetic}}}{N k_{\mathrm{B}}}\,. \tag{7}$$

From statistical mechanics it can be shown that Pressure can be computed from two contributions. One of them is the ideal-gas contribution, the other one comes from the interactions. Without giving details, we just state the formula:

$$P = \frac{1}{V} \left(\frac{1}{3} \sum_{i=0}^{N} m v_i^2 + \sum_{i=0,\, j<i}^{N} \vec{F}_{ij} \cdot \vec{r}_{ij} \right)\,. \tag{8}$$

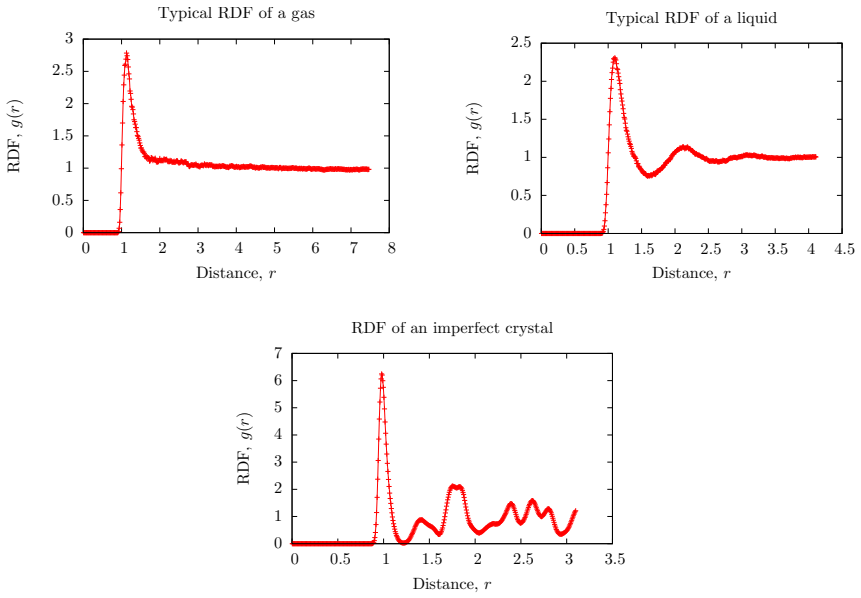

Figure 1: Examples of radial distribution functions obtained from simulations.

An observable containing information about the structure of the system is the of the radial distribution function (RDF) defined as

$$g(r) = \frac{1}{\rho 4\pi r^2 \mathrm{d}r} \sum_{ij} \langle \delta(r - |r_{ij}|) \rangle \tag{9}$$

RDF describes by how many particles a particle is surrounded at a particular distance in comparison with the ideal gas phase, where we have $g(r) = 1$ at every distance r. It can be used to compute all kinds of thermodynamic observables. For our purpose we mention three typical shapes of $g(r)$. In a dilute gas, it is a monotonously decaying function. In a simple liquid it has a form of oscillations which are damped out with increasing r. In an ideal crystal, $g(r)$ contains sharp peaks at well-defined positions which are given by the symmetry of the lattice. Examples of such RDFs are shown in Figure 1.

3.5 Simulating at a desired temperature – velocity rescaling

Because we are using the energy-conserving velocity verlet algorithm and we keep a constant number particles in a box of constant volume, by construction we are simulating the microcanonical $[N, V, E]$ ensemble. However, experiments are typically performed at a given temperature. To be able to compare simulations to experimental data, one would prefer to simulate at a pre-defined temperature, rather than at total energy, i. e. to simulate a canonical $[N, V, T]$ ensemble. One way of driving the simulated system to the desired temperature is rescaling particle velocities. In this procedure, velocities

of all particles are multiplied by a factor such as to achieve the desired temperature (equation 7). This is repeated until the temperature remains constant. The number of steps of velocity rescaling in our program is controlled by the `Nsteps_rescale` variable.

One important problem of the velocity rescaling is that it destroys the Maxwell-Boltzmann distribution of velocities in the system. Therefore, a simulation with velocity rescaling produces neither $[N, V, E]$ nor $[N, V, T]$ ensemble. In the implementation here, the velocity rescaling is only used in equilibration to drive the system to the desired temperature. During the productive simulation run, no velocity rescaling is performed anymore, hence we are simulating an $[N, V, E]$ - ensemble with (average) temperature close to our pre-defined value. In the next tutorial, you will learn about more advanced methods of simulating at a constant temperature.

Task: (1 point)

- *Explain why it is interesting to rescale the velocities, if we want to obtain a constant temperature.*
- *Find which lines in file* `mdloop.c` *are responsible for the velocity scaling. Just provide the line numbers.*

4 Study and analysis of the results

The main point of the this tutorial is to perform a series of simulations, to follow the time-evolution of various observables, to identify the equilibration period and finally to measure the averages of certain obervables. We begin with an $[N, V, E]$ simulation without velocity rescaling.

Task: (3 points)

- *Perform MD simulation at density* $\rho = 0.8$ *with hcp configuration as a starting point and velocity rescaling switched* off. *Set temperature to 1.0.*
- *Plot the time-evolution of total energy to show that it is conserved.*
- *Plot time-evolution of all other observables to illustrate how they relax to constant values.*
- *Identify an observable which relaxes most slowly. From its time-evolution, estimate the duration of equilibration. If it is more than a fraction of the whole simulation, do a longer run.*
- *Plot the velocity distribution and compare it with the Maxwell-Boltzmann distribution function for the given temperature.*
- *Provide all input parameters of all simulations in a table.*

Useful things to note:

- To switch off the velocity rescaling, set `Nsteps_rescale` to zero in the input file.
- Without velocitity rescaling, the `Temperature` parameter from the input file is only used for the initial velocity distribution but the final temperature will be different.
- Zoom the time-evolution plots appropriately, so that the decay can be seen. You do not need to show the evolution of the whole simulation, just the relevant part.

In the next part of the homework, we will use velocity rescaling during equilibration to drive the system to a desired temperature. We will also examine the effect of initial structure on the final properties of the system and compute observables. To obtain correct observable values, it is crucial that the system is properly equilibrated before any measurements.

Task: (2 points)

- *Simulate the same system as you did in the previous case at $T = 1.0$ and switch on the rescaling. If the measured temperature deviates from the desired one by more than 10%, modify the parameters and re-run the simulation.*
- *Perform two more simulations of the same system with different symmetry of the initial configuration.*
- *Plot the time evolution of pressure in all three systems (in separate plots) to show that they differ.*
- *In a table, compare the average values of all observables for all three different systems.*
- *Plot the radial distribution function of all three systems in one plot. Can you conclude that the system at a given density and temperature has the same structure irrespective of the initial configuration?*

Useful things to note:

- From now on it is your responsibility to make sure that your simulations are long enough. Remember that the duration of equlibration is not a universal quantity but depends strongly on system properties.
- After the end of velocity rescaling, the kinetic energy still needs some more time to relax. Therefore always use `Nsteps_rescale < Nsteps_equil`.
- Random configuration may contain enough initial energy to make the sytem explode. Turning on the pre-equilibration by energy minimization may help prevent the explosion.

- Make sure that your total simulation time is much longer than the equilibration, so that enough data are collected.

In the last part of the homework, we will use the same approach as in the previous but we will focus on cases when it may take longer for the systems to equilibrate. Keep energy minimzation turned on to speed things up.

Task: (2 points)

- *Take your system from the previous example and decrease the density to* $\rho = 0.2$*. Identify the duration of equilibration and make the whole simulation 6000 steps longer than equilibration.*
- *Repeat the simulation with different initial conditions and plot the RDF to see if the structures differ. If you increase the number of simulation steps by a factor of 3, will the structures be closer to each other?*
- *Repeat previous steps with the same system at density* $\rho = 1.2$*.*

Optional task: (2 bonus points)

- *Explain, why equilibration takes longer in some cases. In different cases, the reasons may differ!*
- *Explain, why in some cases the structures may differ even though both systems seem to be equilibrated and no further drift of observable values can be seen.*

Optional task: (up to 4 bonus points)

- *If you decide to do this task, first discuss your plans with the tutor.*
- *Add sanity checks for the values of input parameters*
- *Modify the code so that it writes configurations and observables at an interval defined in the input file.*
- *Make the simulation program read and write checkpoints from which the simulation can be continued when desired.*
- *Make the analysis routines into a separate program which reads in the stored configurations and analyzes them.*
- *Any other modification you agree upon with the tutor.*