Worksheet 2: Statistical Mechanics and Molecular Dynamics

Mariano Brito, Keerthi Radhakrishnan, Somesh Kurahatti

November 10, 2023

Institute for Computational Physics, University of Stuttgart

Contents

1 General Remarks

- Deadline for the report is **midnight of 23.11.2023 for groups from Mondays and Tuesdays, midnight of 24.11.2023 for groups from Thursdays**
- On this worksheet, you can achieve a maximum of 20 points.
- The report should be written as though it would be read by a fellow student who attends the lecture, but does not do the tutorials.
- To hand in your report, upload it to ILIAS and make sure to add your team member to your team. If there are any issues with this, please fall back to sending the reports via email
  - Mariano (`mbrito@icp.uni-stuttgart.de`)

  - Keerthi (`keerthirk@icp.uni-stuttgart.de`)
  - Somesh (`skurahatti@icp.uni-stuttgart.de`)
- For the report itself, please use the PDF format (we will *not* accept MS Word doc/docx files!). Include graphs and images into the report.
- If the task is to write a program, please attach the source code of the program, so that we can test it ourselves.
- The report should be 5–10 pages long. We recommend using LaTeX. A good template for a report is available on the home page of the lecture at `https://www2.icp.uni-stuttgart.de/~icp/mediawiki/images/5/52/Report_SHEETNUMBER_NAME1_NAME2.tex`.
- The worksheets are to be solved in groups of two people. We will not accept hand-in-exercises that only have a single name on it.

2 Statistical Mechanics

Task (2 points)

- Consider a system A that consists of subsystems A_1 and A_2, with numbers of possible configurations $\Omega_1 = 10^{31}$ and $\Omega_2 = 10^{28}$, respectively. What is the number of configurations available to the combined system? Also, compute the entropies S, S_1 and S_2.
- By what factor does the number of available configurations increase when $1\,\mathrm{m}^3$ of neon at $1\,\mathrm{atm}$ and $298\,\mathrm{K}$ is allowed to expand by 1% at constant temperature? (Neon should be treated as an ideal gas here.)
- By what factor does the number of available configurations increase when an energy of $100\,\mathrm{kJ}$ is added to a system at initially $T_0 = 400\,\mathrm{K}$ containing $1.0\,\mathrm{mol}$ of particles at constant volume?

(Adapted from: Frenkel, Smit: *Understanding Molecular Simulation*)

Task (3 points)

(**Ideal Gas**) The canonical partition function of an ideal gas consisting of N mono-atomic particles is

$$Z(N,V,T) = \frac{1}{h^{3N}N!}\int \mathrm{d}\Gamma \exp\left[-\beta H\right] \tag{1}$$

in which $\beta = 1/(k_B T)$ is the inverse temperature, the Hamiltonian is given by

$$H = \sum_{i=1}^{N} \frac{\mathbf{p}_i^2}{2m} \tag{2}$$

and $\mathrm{d}\Gamma = \mathrm{d}q_1 \ldots \mathrm{d}q_N \mathrm{d}p_1 \ldots \mathrm{d}p_N$. Derive expressions for the following thermodynamic properties:

- $F(N,V,T)$ (hint: $\ln(N!) \approx N\ln(N) - N$).
- C_V (heat capacity at constant volume) and p (pressure).

(Taken from: Frenkel, Smit: *Understanding Molecular Simulation*)

Task (2 points)

(**Ising Model**) Consider a system of N spins arranged on a simple lattice (1d: a linear chain, 2d: a square lattice, 3d: a simple cubic lattice, etc.). In the presence of a magnetic field, H, the energy of the system is

$$U = -\sum_{i=1}^{N} H s_i - J \sum_{i>j} s_i s_j, \tag{3}$$

where J is called the coupling constant (J>0) and $s_i = \pm 1$. The second summation is a summation over all pairs (d $\times$ N for a periodic system, d is the dimensionality of the system). This system is called the Ising model.

- Show that for positive J, and H=0, the energy of the Ising model is equal to

$$U_0 = -dNJ \tag{4}$$

- Show that the free energy per spin of a 1d Ising model with zero field is equal to

$$\frac{F(\beta, N)}{N} = -\frac{\ln(2\cosh(\beta J))}{\beta} \tag{5}$$

  when $N \to \infty$. The function $\cosh(x)$ is defined as

$$\cosh(x) = \frac{\exp(-x) + \exp(x)}{2}. \tag{6}$$

- Derive equations for the energy and heat capacity of this system.

(Taken from: Frenkel, Smit: *Understanding Molecular Simulation*)

3 Molecular Dynamics: Lennard-Jones Fluid

On this worksheet, you will do Molecular Dynamics simulations as they are commonly used to simulate molecular systems. The system that is to be simulated is a fluid of particles at different densities interacting via the Lennard-Jones potential in two dimensions. Please note that the methods shown on this worksheet are straightforwardly transferable to three dimensions – we choose to limit our simulations to two dimensions only for ease of visualization (and implementation).

All files required for this tutorial can be found in the archive `templates.tar.gz` that can be downloaded from the lecture's homepage.

After you have implemented the Lennard-Jones potential in the task below, you will start with a program very similar to the program that simulated the solar system from the last worksheet, and successively improve the program until it is a fully-fledged Molecular Dynamics simulation program.

3.1 Lennard-Jones Potential

We are going to simulate a system consisting of particles interacting via the Lennard-Jones (LJ) potential V_{LJ}, which has been originally derived for a system of noble gas atoms (see Fig. 1)

$$V_{\mathrm{LJ}}(r) = 4\epsilon \left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right). \tag{7}$$

Here, r denotes the distance between a pair of particles, while σ and ϵ are constants determining the spacial range and energy scale of the potential. On very short distances of the order of the atom radius, the Pauli principle disallows the atoms to overlap, which results in a strongly repulsive potential that should be $\propto e^{-r}$. However, the actual functional form of the core does not really influence the results as long as it is strong enough, therefore the computationally simpler term $\propto r^{-12}$ is used. On intermediate distances, the interaction is slightly attractive, which corresponds to the van-der-Waals-interaction of induced dipoles. It can be shown that this interaction should be $\propto -r^{-6}$, which is the second term in equation 7. For long distances, the interaction is negligible.

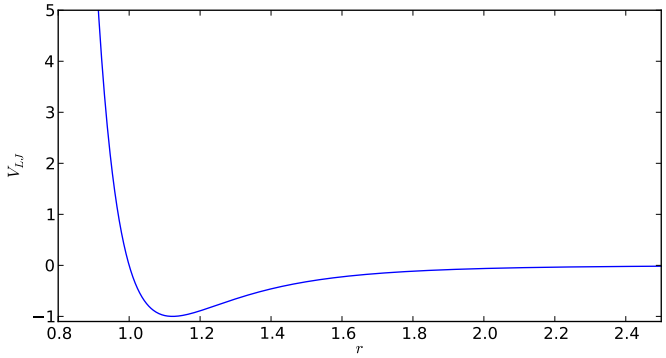

Figure 1: The (12,6)-Lennard-Jones potential ($\sigma = 1$, $\varepsilon = 1$).

3.2 Reduced Units

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

$$U(r) = \varepsilon\phi(\sigma/r)\,, \tag{8}$$

where ε and σ are constants and ϕ is a smooth and differentiable function, it is possible to define a set of dimensionless reduced units such as $x^* = x/\sigma$, $V^* = V/\sigma^3$, $T^* = k_\mathrm{B}T/\varepsilon$ and $p^* = p\sigma^3/\varepsilon$. Within this set of reduced units, all systems interacting with potentials of the form given by equation 8 follow one single universal equation of state. This law is called the *theorem of corresponding states*. Therefore, results of simulations of simple fluids are usually given in these reduced units. One particular implication of the theorem of corresponding states is that we can perform all simulations with $\varepsilon = 1.0$ and $\sigma = 1.0$ and if desired, the results can be transferred 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 quantities on the order of unity.

Therefore, in all the following tasks, we will assume that $\varepsilon = 1$ and $\sigma = 1$.

Hint The archive contains a python script `plot_samples.py`, which demonstrates how to make nice-looking plots. For a more complete introduction to plotting, have a look at `https://matplotlib.org/stable/tutorials/introductory/pyplot.html`.

Task (2 points)

- Implement a Python function `lj_potential(r_ij)` that returns the Lennard-Jones potential (for $\varepsilon = 1$ and $\sigma = 1$) for two particles with the distance vector `r_ij` (which is a 2d-Numpy-array).
- Implement a Python function `lj_force(r_ij)` that returns the Lennard-Jones force (as a 2d-Numpy-array) for two particles with the distance vector `r_ij`.
- Create a distance vector $(d, 0)$ and let d attain 1000 values in the interval $[0.85, 2.5]$. Compute both LJ force and potential for two particles at these distances and create a plot of the potential and the first component of the force against d.
- Implement a fully repulsive potential by shifting the Lennard-Jones potential by an optimum value. Ensure that the cutoff is chosen such that the potential is continuous. Compute both LJ force and potential for two particles at varying distances as in the previous case and create a plot of the potential and the first component of the force against d.

3.3 Lennard-Jones Billiards

A large part of the Python program `ex_3_3.py` from the archive is very similar to the sample solution for simulating the solar system on the last worksheet. It is almost

ready to play 2d-Lennard-Jones billiards. For the program to work it imports the force and potential calculation functions from your solution of the previous task. Therefore make sure that your previous solution is called `ex_3_2.py`. The billiard program sets up five LJ particles ("billiard balls") at certain positions and with certain velocities and simulates them for 20 time units. The only thing that is missing are the functions from the previous task.

The program creates two plots: A plot of the trajectories in the two-dimensional x-y-plane and a plot of the total energy of the system over time. The latter plot can be used to verify that the LJ potential and forces are correct. If they are correct, the energy will be constant, except for a few low-amplitude wiggles when the balls hit each other.

Furthermore, the program creates a file `ljbillards.vtf` in the `VTF`-format. Note that in writing this file, a third dimension is added to be able to use typical visualization programs which usually require 3d-data – setting this dimension to a constant value does not change the simulation result, of course. Open the file in a text editor so that you can see what it contains. The file can also be viewed with the 3d-visualization program VMD[1]. To load the file into VMD, either execute

```
vmd ljbillards.vtf
```

in the shell, or load it via the in-program dialog `VMD Main` → `File` → `New Molecule...`

When you load the file into VMD, the LJ particles are displayed as points which are almost invisible on the screen. To display the particles as colored spheres, open the dialog `VMD Main` → `Graphics` → `Representations...` and choose `Draw Style` → `Drawing Method` → `VDW` and `Draw Style` → `Coloring Method` → `Index`.

VMD loads the whole trajectory of the system. The slider in the `VMD Main` window can be used to watch the system evolve in time.

To create a high-quality image of the currently visible scene, open the dialog `VMD Main` → `File` → `Render`. Choose `Tachyon (internal, in-memory rendering)` and press the button `Start Rendering`. This will create a file `vmdscene.tga` that contains a ray-traced image of the scene. To convert it to a PNG image file, use ImageMagick:

```
convert vmdscene.tga vmdscene.png
```

[1] `http://www.ks.uiuc.edu/Research/vmd/`

Task (2 points)

- Let the program run and visualize the system with VMD.
- Change the position of the fifth particle (index 4) so that it is hit by the third particle (index 2).
- Although the particles almost behave like billiard balls, some of them do not. Which particles and why? Create a plot of the trajectory or a series of images that supports your hypothesis.
- Now introduce 2D boundaries to confine the system. Extend `ex_3_3.py` in order to add four hard walls. They should be placed at $x = 0$, $x =$ `box_l`, $y = 0$ and $y =$ `box_l`, with `box_l` $= 15$. The interaction of the particles with the walls should be elastic. Create a plot of the trajectory. Is the energy of the system still conserved?

Hints for this task

- Write a function `apply_bounce_back(x, v, box_l)` that takes particle positions and velocities and reflect them according to elastic collision when a particle moves out of the squared box of size `box_l`. When should the function be applied?
- Make sure that you initialize the simulation with all the particles inside the box.

3.4 Periodic Boundary Conditions

Our goal on this worksheet is the simulation of a LJ fluid to determine macroscopic bulk properties of the fluid. In practice, however, even the largest modern computers are not able to simulate systems with a macroscopic size with $\approx 10^{23}$ particles, but use systems that are significantly smaller.

One of the problems of fluid simulations with a finite number of particles is how to handle the boundaries of the system. If we would use *open boundary conditions* where the available space is infinite, the particles would simply escape into empty space. Therefore, we somehow have to confine the particles in a finite volume. We could create walls on the boundaries that push back the particles when they try to escape. However, this creates boundary effects, and the properties of the fluid close to the walls differ greatly from the bulk properties that we are actually interested in.

The common solution to this problem is to use *periodic boundary conditions* (PBC), where the system with a finite size is (virtually) multiplied infinitely into all directions. A particle on the right boundary of the system interacts with the particles close to the left boundary of the system as though there would be a copy of the whole system (a *periodic image* of the system). The same goes for all directions. An alternative view

on the situation is that the system lives on a three-dimensional torus. A particle that leaves the system to the right will turn up on the left. This principle is well-known from various computer games.

The advantage of periodic boundary conditions is, that the system is virtually infinite – the environment of the system is the system itself, so there are no boundary effects.

Note that by introducing the periodic boundary conditions, we have also introduced a new parameter of the system, namely the system size L, and we can define the volume of the system $V = L^3$, and the number density $\rho = \frac{N}{V}$, where N is the number of particles in the system.

To visualize a system with PBC, VMD has different functions. These functions can be accessed via the VMD console that you can open by choosing `VMD Main` → `Extensions` → `Tk Console`. This console allows you to type the following commands:

`pbc box` which will draw a box at the boundaries of the system.

`pbc wrap -all` which will wrap all coordinates of the system into the central image. This is only needed when the coordinates can also be outside the central image in the first place.

However, first of all, VMD needs to know about the size of the system. This can be easily done in a VTF file, by adding a line so that it looks like the following:

```
atom 0:1 radius 0.5
pbc 10.0 10.0 10.0
timestep
3.88000009704 2.98000008821 0.0
6.11999990296 5.01999991179 0.0
.
.
```

Long- and Short-Range Interactions

How can we perform simulations with PBC? The trouble is, that there are an infinite number of copies of each particle. If the range of the interactions is not finite, computing the force acting on a single particle becomes very tricky. Such interactions with an infinite range are also called *long-range interactions*, examples for this type of interaction are the Coulomb interaction or gravitation. In fact, there are methods to handle this kind of problem, which you will learn about in the second part of the lecture.

Fortunately, most interactions have a finite range (so-called *short-range interactions*), or at least they converge to zero fast enough so that we can introduce an approximation and truncate the potential at a certain distance (the *maximal interaction range*, or *cutoff*). In that case, interactions with particles that are further away are simply neglected.

In the case of the Lennard-Jones potential, we can define the *truncated LJ potential* as follows:

$$V'_{\mathrm{LJ}}(r) = \begin{cases} V_{\mathrm{LJ}}(r) - V_{\mathrm{LJ}}(r_{\mathrm{cutoff}}) & r \leq r_{\mathrm{cutoff}} \\ 0 & r > r_{\mathrm{cutoff}} \end{cases} . \tag{9}$$

Note that we also shift the potential slightly by $V_{\mathrm{LJ}}(r_{\mathrm{cutoff}})$, so that the potential is continuous at the cutoff. For $r < r_{\mathrm{cutoff}}$ the force due to the truncated LJ potential is the same as due to the full form, otherwise the force is 0. In practice, $r_{\mathrm{cutoff}} = 2.5\,\sigma$ is often used, which we will also do in the rest of this worksheet.

Minimum Image Convention

Now that we know that we use an interaction with a finite range, we can think of how to compute the forces on the particles. The so-called *minimum image convention* states that if the maximal interaction range is less than the system size L, to compute the force that a particle B exerts on particle A, you only have to compute the interaction with a single periodic image of particle B, the *minimum image* that is closest to the particle A. This minimum image of particle B is not necessarily in the same image of the system that particle A is in!

Knowing this, it becomes significantly easier to implement periodic boundary conditions. In particular, the code of the LJ billiard from the previous task has to be changed only at a few places to implement periodic boundary conditions. There are several possibilities to implement PBC.

Note that it is *not* necessary to keep the particles in the central image, by folding them back when they leave the central image. Instead, one can also simply let the particles propagate outside of the central image and take into account the PBC only when computing the interactions. Doing so has a few advantages when it comes to computing observables. For visualization, coordinates that are folded back into the central image are more useful. Therefore, VMD supports the function `pbc wrap` that folds all coordinates into the central image.

Task (2 points)

- Make a copy of the solution program from the previous task (`ex_3_3.py`) and call it `ex_3_4.py`. Modify the LJ potential and force such that it implements the truncated LJ potential (with $r_{\text{cutoff}} = 2.5$).
- Extend the program `ex_3_4.py` such that it takes into account periodic boundary conditions.
- Which functions need to be modified to implement them?
- To test the PBC, set up two particles in a system with edge length $L = 10.0$, initial positions $\mathbf{x}_0 = (3.9, 3)$ and $\mathbf{x}_1 = (6.1, 5)$, and initial velocities $\mathbf{v}_0 = (-2, -2)$ and $\mathbf{v}_1 = (2, 2)$. Simulate them for 20 time units. With PBC, the particles should collide. Plot the trajectory.

3.5 Lennard-Jones Fluid

Now we are ready to simulate a Lennard-Jones fluid!

The Python program `ex_3_5.py` from the archive is an almost complete simulation program that should be very similar to your LJ billiard program with PBC.

To simulate a LJ fluid, we simply have to increase the number of particles and increase density. When the number of particles becomes large, setting up the system manually by defining the particles' coordinates is unpractical. On the other hand, if we put in the particles randomly, chances are that some of the particles will overlap (in particular at high densities), which would lead to extremely high forces possibly leading to an explosion of the system. In our case, we therefore simply set up the particles on a 2d square lattice with n sites per dimension, resulting in a lattice with $N = n^2$ sites.

The parameters of this program therefore are not the number of particles N and the system size L, but the number of particles per dimension of the cubic lattice n and the density ρ.

The program itself measures the run time of the integration loop using the Python module `time` and prints it to the standard output.

Task (3 points)

- Extend the program `ex_3_5.py` to set up the particles on a cubic 2d-lattice with a given number of particles per dimension n at a given density ρ.
- Perform simulations of the LJ fluid with $n = \{3, 4, 5, \ldots, 12, 13\}$ particles per dimension of the quadratic lattice (time step $\Delta t = 0.01$, end time $t_{\max} = 1.0$, density $\rho = 0.7$) and measure their run times.
  Plot the measured run times over the respective total particle number. Produce not only a linear plot but also semilog and loglog plots. On basis of these plots discuss the general scaling behavior.
- Based on your insights on the general scaling choose an appropriate fit function and explain your reasoning. Fit this function to your data and add the fit to your plots. In your report, give both functional form and values of the fit parameters. Discuss the run time scaling behavior based on the results of your fit.

Verlet lists

In the lecture, you have learned about several ways to speed up naive MD implementations like the LJ fluid of the previous task. Here, you will have a look at one of these methods, namely Verlet lists, which are a data structure to efficiently maintain a list of all particles within a given cutoff plus a skin (for more detail please have a look at the lecture notes).

The Python program `ex_3_6.py` is an almost complete implementation of the 2d LJ fluid of the previous tasks, missing just a few lines of the implementation of the Verlet lists.

Task (4 points)

- In the program the actual creation of the Verlet lists is missing. Implement the Verlet list creation function `get_verlet_list(...)`.
- Why does the function return a copy of the current particle position in addition to the actual Verlet list?
- Measure the run time of the integration loop as done in the previous task. Also count how often the Verlet lists are updated during the simulation. Print out both run time and number of updates.
  Now perform simulations for a constant number of particles per dimension $n = 8$ while varying the skin $= \{0.0, 0.1, 0.2, \ldots, 0.9\}$. Plot both run time and number of updates over the skin and discuss you observations.
- Choose the skin with the smallest run time from the previous task. Adapt the simulation time `T_MAX` to match the one used in task 3.5. Repeat the run time measurements of task 3.5 with the Verlet list program and plot the results together with the measurements of 3.5. Compare and discuss the different scaling behavior. What scaling behavior would you expect using Verlet lists?