RWTH Aachen INSTITUTE FOR THEORETICAL SOLID STATE PHYSICS

Computational Physics (SS 2020)

Prof. Dr. Riccardo Mazzarello **Sheet 5 (40 points)**

Hand-in date: 29 May, 2020, 18:00 p.m.

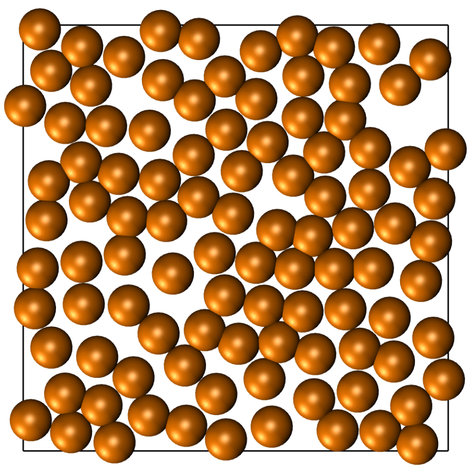
New remarks worth reading

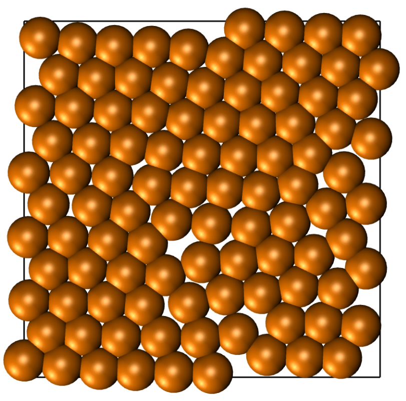
- Please hand in your solutions in teams of two students.
- Provide a *single* PDF-file containing the written solutions of all exercises. If it contains a scan, please make sure it is well readable. Write your names on the first page.

  Plots must be embedded into this file. Plots given separately as additional files are not accepted.
- Use a current version of Python 3 for your implementations. Write readable and *commented* code with self-explanatory variable names.
- Please hand in your solutions, i. e., **only the PDF-file (containing plots and answers) and the source code files, but no trajectory files**, as a single ZIP-file named

  Surname1-MatNum1-Surname2-MatNum2-SheetX.zip,

  where MatNum is your matriculation number, by uploading it to RWTH Gigamove (`https://gigamove.rz.rwth-aachen.de`). Please send the download link to **manconi@physik.rwth-aachen.de** making sure it is still valid on June 15, 2020.





Building a Molecular Dynamics Engine (20 points)

This first part consists in implementing a molecular dynamics engine that will enable you to produce the trajectories needed for the analysis in the second task. Since the analysis will be about a two-dimensional system with a square simulation box, it is okay to write the engine explicitly for this case. However, we encourage you to write general code that can be called for arbitrary initial configurations, simulation-box sizes and simulation parameters. A possible way to structure such an engine could be a Python class containing all information about the system as well as methods to initialize the system, run the simulation, and write the trajectory to the disk.

Stick to the following points:

- Velocities are initialized by assigning random numbers according to the Maxwell-Boltzmann distribution (hint: `numpy.random.normal`). Make sure you have no center-of-mass movement. (4 points)
- Use the velocity Verlet method to integrate the equations of motion. (2 points)
- During the generation of a trajectory, do not wrap atomic coordinates back into the original simulation box when they leave it.
- The particles shall interact via a Lennard-Jones pair-potential. (4 points)
- Instead of using a cutoff distance when computing interactions, apply the minimum image convention: To compute the force acting on particle i due to particle j and its infinitely many image particles, consider only that periodic image of particle j which is closest to particle i. (8 points)
- Your code should be able to perform NVT simulations for equilibration and NVE simulations for production. For the former, use velocity rescaling every 10 MD steps to arrive at the target temperature (note the number of degrees of freedom of the atoms). (2 points)

Some further hints and strong suggestions:

- Divide the problem into many small functions (or class methods) that each perform a small, specific subtask. Test every portion of your code to make sure it is working correctly.
- As it takes some time to run the simulation, do not produce and analyze a trajectory in the same Python script. Rather, let your MD engine write the trajectory and important quantities to files and let your analysis scripts read in the data from these files. One file could contain, with one MD step per line, the number of the time step (integer), the time, the kinetic, potential and total energy, and the temperature. A second file could contain the atomic positions and a third one the velocities. To save space and reduce the runtime, you can choose to write to disk (and then analyze) only, e. g., every 10th frame (time step) as far as positions and velocities are concerned (but store energies and temperature for *every* time step).
- As a production run starts from the final state of the preceding equilibration run, it is helpful to implement the possibility to restart the simulation, i. e., to initialize the positions and velocities based on the last frame of existing trajectory files.

Let the Atoms Dance (20 points)

Setting

Consider a two-dimensional gas of Argon atoms interacting via a Lennard-Jones potential. The initial atomic configuration is a square lattice. We use a square simulation cell containing $N = 10 \times 10 = 100$ atoms and employ periodic boundary conditions. The Lennard-Jones parameters for Argon are given by $\epsilon = 1.65 \times 10^{-21}\,\mathrm{J}$ and $\sigma = 3.4 \times 10^{-10}\,\mathrm{m}$, and the mass of an Argon atom is 39.9 u. Two different atomic densities shall be examined in this task: $\rho_1 = 0.07\,\text{Å}^{-2}$ and $\rho_2 = 0.1\,\text{Å}^{-2}$. The simulations shall be performed with a time step of $\tau = 0.01\,\mathrm{ps}$.

Units (2 points)

Use the following units throughout this exercise and in your code: [Length] = 1 Å, [Time] = 1 ps, [Mass] = 1 u, [Temperature] = 1 K.

- Which values do ϵ, σ, and the Boltzmann constant k_B have in these units?
- How large are the sides of the simulation box for the two densities given?

Analysis (18 points)

- For each of the two densities, perform an MD simulation at $T = 150\,\mathrm{K}$ consisting of the initialization, equilibration, and production.
- Obtain from your simulations the evolution of the total energy and temperature. Plot these quantities over time and interpret the results. How long do the systems need to equilibrate? (At this point, make sure that your equilibration has been chosen long enough.) In the NVE production run, it is okay to have a small deviation of the average temperature from 150 K. (4 points)
- Provide a visual snapshot of each of the two systems after equilibration, i. e., a picture of the square cell with all 100 atoms. For this visualization, wrap the atoms into the original simulation box. Describe and compare the resulting structures. To generate snapshots, you can use some Python library, such as `matplotlib`, or the more sophisticated VMD software, available at `https://www.ks.uiuc.edu/Research/vmd/` (note that the download requires registration) . (4 points)

For the following tasks, only use the production parts of your trajectories. Have a look at your resulting plots and make sure that the data is converged. If it is not, choose longer simulation times to collect more frames.

- For each system, compute from the velocities the speed distribution, i. e., the distribution of the magnitude $|\vec{v}|$. Take into account all atoms and enough frames. Plot a histogram of $|\vec{v}|$ and the 2D Maxwell-Boltzmann distribution, and compare the results. How many frames did you use for the plot? (4 points)
- Compute from the positions the radial pair-correlation function $g(r)$. Take into account all atoms and enough frames, and check for correct normalization. Does $g(r)$ behave as expected for large r? Plot $g(r)$ for ρ_1 and ρ_2 in the same plot. Interpret and explain the results. How many frames did you use for the plot? (6 points)