Lab 4
Handout: Classical Molecular Dynamics of Silver

Giuliana Materzanini[1] and Leonid Kahle[2]
Theory and Simulation of Materials (THEOS),
École Polytechnique Fédérale de Lausanne.

[1] giuliana.materzanini@epfl.ch
[2] leonid.kahle@epfl.ch

May 2018

1 Molecular dynamics codes and references

In this lab, we will be using the General Utility Lattice Program (GULP) package to run the classical molecular dynamics (MD) simulations. You have already experienced some of the features of GULP at the LAB 1, the difference will be that we will move from static calculations to dynamical calculations. It is also a very good practice to always consult the manual of the code, it can answer a lot of your question:

- PDF version: `https://nanochemistry.curtin.edu.au/local/docs/gulp/gulp4.0_manual.pdf`
- HTML version: `https://nanochemistry.curtin.edu.au/local/docs/gulp/help_40_txt.html`

The manual provides the tools to setup the input files and describes the information contained in the output files. It is also a good resource for learning about classical MD method, as it explains the methods used and the potentials involved in the calculations. Some other useful links about the MD simulations can be used as good references, such as:

- `http://cms.sjtu.edu.cn/doc/reading/md/A_Molecular_Dynamics_Primer_(Ercolessi).pdf`
  A brief introduction to the MD technique and interatomic potentials.
- `https://github.com/itamblyn/FORTRAN/blob/master/md/mdx/md1.f90`
  An example of an MD code, written in Fortran 90.
- `http://www.materialsdesign.com/appnote/embedded-atom-method-simulation`
  An explanation of the embedded atom method, that will be applied in our simulations for silver.

It is important to note that beside of GULP there are also several MD codes available for public use, such as: LAMMPS, CHARMM, AMBER, NAMD, GROMACS, MOSCITO, and DL_POLY, etc. Many of these are listed on this webpage:

- `https://en.wikipedia.org/wiki/List_of_software_for_molecular_mechanics_modeling`

We also suggest two books that are standard references to learn about MD simulations:

- Frenkel & Smit, Understanding Molecular Simulations, Academic Press (2002). In particular Chapter 4 gives a detailed description of the basics of MD, we recommend you to read this part. A good description of the various ensembles is provided in Chapter 6. A nice explanation of the methods to treat electrostatic interactions is reported in the first part of Chapter 12, whereas the algorithms to speed up the calculation of non-bonded interactions are reported in Appendix F.
  Science Direct link

- Allen & Tildesley, Computer Simulations of Liquids, Oxford (1987). This text is more formal than the previous one, but it also contains some important derivations and formulas. Green-Kubo relations and other quantities important in MD runs are well described in Chapter 2. The expression of the errors in statistical averages is quite clear and can be found in Chapter 6.4.
  Google Books link

2 Quick summary of molecular dynamics

In the MD technique the time evolution of a set of N interacting atoms is followed by integrating the set of classical (Newton's) equations of motion

$$\vec{F}_i = m_i \vec{a}_i, \tag{1}$$

where i labels the i^{th} atom in the system, m_i is the atom mass, $a_i = d^2\vec{r}_i/dt^2$ its acceleration, and $\vec{F}_i$ the force that acts upon it, due to its interaction with the other atoms. In principle from any set of initial positions and velocities, the positions $\{\vec{r}_i\}$, velocities $\{\vec{v}_i\}$ at each MD time step will be determined by knowing the forces and thus the acceleration. The force on each atom i, $\vec{F}_i$, is the negative of the derivative of the potential energy with respect to the position of the atom

$$\vec{F}_i = -\nabla_{\vec{r}_i} U \tag{2}$$

where, $U = U(\vec{r}_1, ..., \vec{r}_N)$ is the potential energy of the system. For simple systems (for example the harmonic oscillator), the equation of motion can be integrated analytically, but for more complex systems we integrate numerically using a finite timestep, which can eventually cause the computed trajectory to deviate from the true trajectory.

MD can therefore be seen as a technique to follow the *time evolution* of a system: the trajectory of the N atoms (i.e. the set of the 2 N vectors $\{\vec{r}_i\}$, $\{\vec{v}_i\}$) is determined from

the initial set of positions and velocities by numerically integrating the equations of motion (once given a form for the potential U).

However, one can also use MD as a *statistical mechanics method*, which is the more common usecase and also what this lab is about. This means that we are not interested in the trajectory itself, but as a set of configurations distributed according to a statistical ensemble (i.e. to some distribution function). For instance, if we place the N atoms in a box with constant volume V and we keep their total energy E (kinetic + potential) constant during the simulation, it follows automatically that each time step of the system trajectory will be a point in the *microcanonical* space phase (N, V, E) at those constant N, V, E values. Statistical mechanics calculates thermodynamic variables as averages of the corresponding (microscopic) dynamical variables over the phase space (i.e. the $6N$ coordinates and momenta) multiplied by the distribution function appropriate of that ensemble. Such an average, indicated by the angular brackets $\langle\rangle$, is indeed called *ensemble average*. Below we provide the ensemble averages in the microcanonical (NVE) and canonical (NVT) ensembles of a quantity A:

$$\langle A \rangle_{N,V,E} = \frac{\int A(\vec{r},\vec{p})\delta(\mathcal{H}(\vec{r},\vec{p}) - E)d\vec{r}d\vec{p}}{\int \delta(\mathcal{H}(\vec{r},\vec{p}) - E)d\vec{r}d\vec{p}} \tag{3}$$

$$\langle A \rangle_{N,V,T} = \frac{\int A(\vec{r},\vec{p})e^{-\beta\mathcal{H}(\vec{r},\vec{p})}d\vec{r}d\vec{p}}{\int e^{-\beta\mathcal{H}(\vec{r},\vec{p})}d\vec{r}d\vec{p}} \tag{4}$$

where $\mathcal{H}$ is the Hamiltonian used and $A(\vec{r},\vec{p})$ is the value of A at the point $\vec{r},\vec{p}$ in phase space. Besides, according to the *ergodic hypothesis*, if we let evolve a system in time, after a reasonable lapse of time it will explore different regions of the phase space according to their ensemble distribution function. This means that, after a long enough evolution time, the temporal average along the trajectory will be equal to the ensemble average over the phase space, and so we can use our trajectory as a sequence of "samples" to be averaged in time to give the macroscopic thermodynamic quantity of interest:

$$\bar{A} = \frac{1}{T}\int A(\vec{r}(t)\vec{p}(t))dt = \langle A \rangle \tag{5}$$

From now on we will use the curly brackets $\langle\rangle$ simply to mean a time integral over our trajectory, as implied by the ergodic hypothesis.
According to these considerations, any MD code will look like the following pseudo-algorithm which include the five main steps:

```
program md
    call initialize
    do
        call compute_energy_and_forces
        call integrate_equation_of_motion
        call sample_averages
        if (converged_averages .or. exceed_time) exit
    end do
end program md
```

Thus, we need to: 1) Initialize positions and velocities, 2) calculate the energies and forces, 3) integrate the equations of motion, 4) sample averages and 5) check the convergence of the average quantities or wall time. In the next sections we will briefly explain those steps, to get you familiar with the classical MD. We advice you to consult the provided references and, in case you have doubts, ask the tutors.

2.1 Initialization positions and velocities

The MD codes that deal with solids and liquids have to employ periodic boundary conditions (PBC). But, it is not mandatory, if you work with molecules you can turn the PBC off. Hence, using PBC the cell vectors and starting positions are explicitly provided in the input file. As an alternative, an ordered crystal structure can be chosen as starting point. Velocities can be given in the input (for example, in the case of restarting from a previous run) or can be randomly generated according to the Maxwell-Boltzmann or another kind of distribution (when starting a new run).

Both the initial positions and the initial velocities can be quite far from the equilibrium at the beginning of the simulation. Thus, except when restarting a previous equilibrated simulation, an equilibration run should be performed to bring the system to the desired thermodynamic state. The length of the equilibration run depends on how far the starting state of the system is from the required one, also, it is very system dependent. For example, to get liquid water at 350 K will require a much shorter equilibration time by starting an equilibration run from a configuration of liquid water obtained at 300 K, than it would by starting from ideal crystalline ice. Effective achievement of the final thermodynamic state can be difficult to detect, especially when dealing with slow, non-ergodic systems, such as liquids at low temperatures or amorphous solids. In these cases, a careful estimate of the error in the averages of the thermodynamic quantities of interest (e.g., temperature, volume, pressure, etc.) should be performed.

2.2 Empirical energy and forces

Calculation of energy and forces depends on the choice of the interaction potential. In our LAB 4 exercises we will employ the embedded atom method (EAM), that performs better for metals: giving good cohesion and ground-state impurity energies, reasonable binding energies of hydrogen to vacancies as well as correct hydrogen adsorption sites and migration energies, compared to experiments (check the original paper for more details: PRB 29, 6443, 1984).

The use of the EAM potential for metals is needed because of the nature of the metallic bonds. The free electron-like behavior cannot be easily captured by the standard pair potentials with bonded, non-bonded and electrostatic parts, such as Lennard-Jones potentials. The EAM potential consists in the following formula:

$$U_i = F_\alpha \left(\sum_{j \neq i} \rho_\beta \left(r_{ij} \right) \right) + \frac{1}{2} \sum_{j \neq i} \phi_{\alpha\beta} \left(r_{ij} \right), \tag{6}$$

where: U_i is the potential energy of an atom i, F_α is the so called embedding energy, which is

the energy required to place an atom i of type α into the electron cloud, ρ_β is the contribution to the electronic density of the atom j of type β at the location of atom i, $\phi_{\alpha\beta}$ is a short-range pair potential part, which depends on the distance r_{ij} between atoms i and j. As one can see in the GULP manual, there are several different functional forms for ρ and F, and also a modified version of the EAM potential, called modified embedded atom method (MEAM), which considers aspherical deformations for the atoms. Further, one can derive some ground-state properties from this potential energy definition, such as, elastic constants, sublimation and vacancy formation energies. Hence, one can define the F and ρ functions in an empirical way, although many potentials are fitted to higher precision quantum calculations, such as the density functional theory.

In summary, the EAM and MEAM potentials are composed by a functional of the sum of the atomic charges (spherical or aspherical), plus a short-range pair potential. This way the free electron-like behavior of the metallic bonds can be well described. Moreover, in order to save computational time, in the short range potential one usually includes only the interactions between an atom and its nearest neighbors (defined as the atoms laying within a defined cutoff distance from the atom itself). This list of neighbors can be updated after a defined amount of time steps, depending on the system considered, which can save CPU time significantly.

2.3 Integration of the equations of motion

Newton's equation of motion can be integrated numerically by means of several different integration algorithms. Preservation of the phase space volume and time reversibility are important features of some of the most commonly used integrators, such as the velocity Verlet and the leap-frog algorithm. Nonetheless, in order to speed up the calculations some less accurate schemes can be adopted. The Verlet algorithm is very well described in the Allen & Tildesley book, it consists of simple Taylor series expansions around $t + \Delta t$ and $t - \Delta t$ summed up, giving a local error on the atomic positions of the order of Δt^4 per integration step.

In order to simulate a system in the canonical (NVT) ensemble, a thermostat is added to the equations of motion. Some of the most commonly used thermostats include velocity rescaling, Berendsen and Nose-Hoover thermostats. Again, the interested reader can find this is Allen & Tildesley book. It should be noted that some of these thermostats require the tuning of their frequency, which is related to how strongly the system would be coupled to the thermostat. If the coupling is too strong, the system will thermalize fast, but quantities will be affected by the thermostat. If the coupling is very weak, the system will not reach the desired temperature.

2.4 Sample averaged quantities

MD is a tool for computing thermodynamic quantities from time averages under the assumption of ergodicity (see above). As the system visits different regions of the configuration space, most of the quantities that are computed in a MD run fluctuate. The average of these quantities over the entire simulation will provide an estimate of the thermodynamic quantity of interest. Even though the ergodic theorem is valid only for infinitely long MD simulations,

a simulation can be stopped when the mean has converged. Suppose we are analyzing a tape of simulation results that contains a total of t_{run} time steps, or configurations. The run average of some property A is:

$$\langle A \rangle_{run} = \frac{1}{t_{run}} \sum_{t=1}^{t_{run}} A(t). \tag{7}$$

and its variance is:

$$\sigma_A^2 = \frac{1}{t_{run}} \sum_{t=1}^{t_{run}} \left(A(t) - \langle A \rangle_{run} \right)^2 \tag{8}$$

Looking at the variance (or, synonymously, the fluctuations) of a quantity A is very important in some cases. Firstly, fluctuations inform you how much you can trust your results:

- We will see in the lab how the fluctuations in the temperature in a microcanonical run scale with system size (i.e. the number of atoms). In such a case, large fluctuations are an indication that you are very far from the thermodynamic limit.
- Other kinds of fluctuations have a direct physical interpretation: the specific heat capacity can be directly computed from fluctuations in the total energy in the canonical ensemble.

Also, a quick note on error estimates in MD (but this is not needed for the assignment): If we were to assume that each quantity $A(t)$ is independent of the others, then the error of the mean would be given by:

$$s(\langle A \rangle_{run}) = \sqrt{\sigma^2(A)/t_{run}} \tag{9}$$

But in practice we have to assume that successive configurations are correlated up to the autocorrelation time of the quantity A, t_A This means that the number of measures should be reduced to $t_{run}/2t_A$: Thus, the error in the mean is given by:

$$s(\langle A \rangle) = \sqrt{\frac{2t_A}{t_{run}}} \sigma_A. \tag{10}$$

See Allen & Tildesley book (Chapter 6) for an extensive explanation.

3 GULP getting started

3.1 Before you begin any calculation

Compared to the *ab-initio* calculations in LAB 2 and LAB 3, the classical MD simulations in this lab are faster and require less memory. Therefore we don't need to use here the high-performing supercomputers resource. As previous labs, we provide a `LAB4.zip` file which contains an example of how to run the MD simulation with the GULP. Unzip the folder and port it into the virtual machine we provide.

3.2 GULP input data parameters

Now you will perform a test run of GULP. In order to do that, you first create the directory `Test` and then copy the input file that we provided to that directory: We assume you have LAB4 in your home folder.

```
theos@theosvm:~/LAB4$ mkdir Test
theos@theosvm:~/LAB4$ cd Test
theos@theosvm:~/LAB4/Test$ cp -r Examples/input_md_sample.gin md_test.in
```

The file `input_md_test.gin` is an example of input file that contains the information needed to compute a MD run at constant number of particles, volume and energy (NVE). You can view the file with the command:

```
theos@theosvm:~/LAB4/Test$ less md_test.gin
```

it will look like this:

```
conv md
title
molecular dynamics of silver
end
cell
    4.085    4.085     4.085    90 90 90
fractional    4
Ag     0.000000000     0.000000000     0.000000000
Ag     0.000000000     0.500000000     0.500000000
Ag     0.500000000     0.000000000     0.500000000
Ag     0.500000000     0.500000000     0.000000000
#
#  modified version of the Cleri-Rosato potential for GULP (Ag)
#  from F. Cleri and V. Rosato
#  Phys. Rev. B, 48, 22 (1993)
#
#  This potential uses the Embedded Atom Model
#
species
Ag core  0.000
manybody
Ag core Ag core  0.0 8.0
#
#  Functional
#
eam_functional square_root
Ag core  1.000
#
#  Density term
```

```
#
eam_density exponential 0
Ag core  1.387684 2.173423 2.888531
#
#  Repulsive two-body components -
#  second power is a dummy argument
#
buckingham
Ag core Ag core    11454.950282 0.264324 0.0 0.0 12.0
#
# MD specific parameters
#
supercell 2 2 2
ensemble nve
temperature 1000
integrator leapfrog verlet
equilibration 1.0 ps
production   2.0 ps
timestep     0.001 ps
sample       0.02 ps
write        0.02 ps
output trajectory ascii Trajectory.trg
dump every 100 Recovery.res
```

Below you will find a brief explanation for each of the input parameters.
For an exhaustive explanation of the physical meanings of these parameters please refer to the pdf version of the GULP manual, whereas you can find the same input parameters in the html GULP help file or by clicking on the links that we provide for each of them.

- line 1

  ```
  conv md
  ```

  To specify that you are performing a MD calculation at constant volume (fixed cell size).
  See manual link for conv and manual link for md.

- lines 2-4

  ```
  title
  molecular dynamics of silver
  end
  ```

  A user chosen title, in order to organize your calculations.
  See manual link for title.

- lines 5-6

```
cell
...
```

  Lattice parameters of the unit cell: these are 6 numbers - 3 lengths a, b and c, given in Angstroms, and 3 angles, α, β, γ in degrees - specifying the edges and the angles of the unit cell. For solids (like in the case of Ag considered here) this is the crystallographic unit cell. For gases and liquids, these parmeters simply specify the shape of the box containing the atoms or the molecules.
  See manual link for cell.

- lines 7-11

```
fractional 4
...
```

  Number of atoms in the simulation cell, atomic species and atomic positions in crystallographic (fractional) coordinates.
  See manual link for fractional.

- lines 12-18 Here the section with details on the potential used starts.

- lines 19-20

```
species
...
```

  Atomic species, atomic type (core/shell) and atomic charge.
  See manual link for species.

- lines 21-22 specify that a manybody potential acts between the defined pair of atoms and give its parameters (atom1 atom2 r_{min} r_{max}).
  See manual link for manybody.

- lines 26-27

```
eam_functional square_root
...
```

  Definition of the functional part of the EAM potential with its parameters. In this case we have a "square root" functional (see also the pdf GULP manual).
  See manual link for eam functional.

- lines 31-32 define the functional form of the density (here "exponential") and its parameters.
  See manual link to eam density.

- lines 37-38 define the short-range two-body potential, that here is a Buckingham potential, and its parameters.
  See manual link to buckingham.

- lines 39-41
  Here the section with details on the molecular dynamics simulation run starts.

- line 42

  ```
  supercell 2 2 2
  ```

  Supercell size: the simulated system will have a cell that is $2 \times 2 \times 2$ times the unit cell. Since fluctuations of thermodynamic quantities decrease with the system size, we can improve accuracy with bigger cells. Unfortunately, large system sizes require a longer computational time, and a compromise between size and time is mandatory. The choice of the cell can be crucial when dealing with long wavelength perturbations or with phase transitions.
  See manual link for supercell.

- line 43

  ```
  ensemble nve
  ```

  To define the thermodynamic ensemble of the calculation (manual link for ensemble), in this case constant number of particles, volume and energy (microcanonical, NVE). For a canonical ensemble (NVT), beside `nvt` there is the Nose parameter, so that this line will look like

  ```
  ensemble nvt 0.01
  ```

  The Nose parameter (here 0.01) is the frequency of the Nose thermostat, that is usually set up to the typical vibration of the system (in this case silver).

- line 44

  ```
  temperature 300
  ```

  To define the simulation temperature (manual link for temperature).

- line 45

  ```
  integrator leapfrog verlet
  ```

  To define the method to integrate the equations of motion numerically (manual link for integrator).

- line 46

```
equilibration 1.0 ps
```

To define the total equilibration time of the simulation (manual link for equilibration).

- line 47

```
production 2.0 ps
```

To define the total production time of the simulation (manual link for production .

- line 48

```
timestep 0.005 ps
```

To define of the timestep for the integrator (manual link for timestep .

- line 49

```
sample 0.050 ps
```

To control how often the properties of the molecular dynamics run are to be sampled and output to the standard output channel. Averaged properties are also based on these samples (manual link for sample).

- line 50

```
write 0.050 ps
```

To control how often the coordinates (or other chosen quantities) are written to the trajectory files (manual link for write).

- line 51

```
output trajectory ascii Trajectories.trg
```

Choice of the format and file name of the trajectory file (manual link for output). Other quantities like the pressure tensor, phonon density of states and so on.

- line 52

```
dump every 1 Recovery.res
```

Frequency that the restart file is written (manual link for dump).

3.3 Running the classical molecular dynamics

Now that you already know how to execute GULP, you will have to use a slightly different input file to run a molecular dynamics calculation instead of the static total-energy calculations that you have run before. A sample run of GULP can be done as follows:

```
theos@theosvm:~/LAB4/Test$ gulp < md_test.gin > md_test.gout
```

where the optional `&` lets you keep control of the terminal while the job is running.

Once the program has been executed, a few files are written. The `md_test.gout` file contains all the details of the input files and of the run itself. Two more files are generated, that are useful to restart a simulation when the job crashes or when you want to improve the statistics of your calculations: (i) one with the trajectories (`.trg`) and (ii) a restart file (`.res`) containing the final state. This allows us to run a new simulation starting from the last frame of a previous run. To restart the calculation, first we save the input and output files in the first run:

```
theos@theosvm:~/LAB4/Test$ cp md_test.in md_test_old.in
theos@theosvm:~/LAB4/Test$ cp md_test.out md_test_old.out
theos@theosvm:~/LAB4/Test$ cp Trajectories.trg Trajectories_old.trg
theos@theosvm:~/LAB4/Test$ cp Recovery.res Recovery_old.res
```

Then, the restart file is just an input file with different positions and velocities, and we can use for a new run:

```
theos@theosvm:~/LAB4/Test$ cp Recovery.res md_test.gin
```

Next, we can modify the new `md_test.gin` file to increase the duration of the MD run, in order to get a better statistics. At the same time we can introduce a thermostat if needed, i.e. run in an NVT ensemble (remember to re-equilibriate if you change the parameters, before gathering any new statistics). To analyze the two sets of trajectories together we must concatenate the files in the correct order, as in the example below (here it is in two steps for simplicity):

```
theos@theosvm:~/LAB4/Test$ tail -n +3 Trajectories.trg > temp
theos@theosvm:~/LAB4/Test$ cat Trajectories_old.trg temp > \
Trajectories_combined.trg
```

3.4 Using bash scripts to run GULP multiple times

In order to check the numerical convergence of your calculations with respect to MD time-step, supercell size etc, you will have to run GULP multiple times with different values for those parameters. Of course you may create a new input file for every set of parameters, but using `bash` scripts can help you to speed up, automatize you work. In this lab we provide you (in `/LAB4/EX1` directory) with an useful `bash` script, that we copy below, which automatically loops over the size of the cell, temperature, equilibration time and timestep values. You can also write a `bash` script by yourself, or modify this one according to your own taste. The script looks like this:

```
#!/bin/bash -f

# Script written to run GULP multiple times on one node.
# The script can loop over 4 different input parameters:
### the time step, specified in list_time_steps
### The temperature, specified in list_temperatures
### The supercell size, specified in list_supercell_size
### The equilibration time, specified in list_equilibration_time

# There are several ways to set your list in BASH.
### For explicit definition of e.g. your timesteps, you can do
##### list_time_steps="0.001 0.003 0.01" #######

### You can also you the seq command to create a sequence as in:
##### list_time_steps=‘seq 0.001 0.001 0.01‘
### This creates a sequence of values between 0.001 and 0.01, every 0.001

list_time_steps="0.001"
list_temperatures="1000"
list_supercell_size="2"


# This is the executable path for gulp
exec=‘which gulp‘

# This is the executable path for the parser
parser="python ~/LAB4/scripts/parser.py"

# These are constants that the script does not loop over, for production time
# and sampling stepsize
prod="5.0"
sample="0.02"
equilibration="5.0"
# Start loops
for cellsize in $list_supercell_size; do
    for temperature in $list_temperatures; do
            for time_step in $list_time_steps; do
echo "Cell size ${cellsize} Temperature ${temperature}, Timestep $time_step ps"
                base_name="${cellsize}_${temperature}_${time_step}"
                cat > md_test_${base_name}.gin << EOF
conv md
title
molecular dynamics of silver
end
cell
```

```
    4.085    4.085     4.085    90 90 90
fractional    4
Ag     0.000000000     0.000000000     0.000000000
Ag     0.000000000     0.500000000     0.500000000
Ag     0.500000000     0.000000000     0.500000000
Ag     0.500000000     0.500000000     0.000000000
#
#  modified version of the Cleri-Rosato potential for GULP (Ag)
#  from F. Cleri and V. Rosato
#  Phys. Rev. B, 48, 22 (1993)
#
#  This potential uses the Embedded Atom Model
#
species
Ag core  0.000
manybody
Ag core Ag core  0.0 8.0
#
#  Functional
#
eam_functional square_root
Ag core  1.000
#
#  Density term
#
eam_density exponential 0
Ag core  1.387684 2.173423 2.888531
#
#  Repulsive two-body components -
#  second power is a dummy argument
#
buckingham
Ag core Ag core    11454.950282 0.264324 0.0 0.0 12.0
#
# MD specific parameters
#
supercell ${cellsize} ${cellsize} ${cellsize}
ensemble nve
temperature ${temperature}
integrator leapfrog verlet
equilbration ${equilibration} ps
production   ${prod} ps
timestep     ${time_step} ps
sample       ${sample} ps
write        ${sample} ps
```

```
output trajectory ascii Trajectories_${base_name}.trg
dump every 100 Recovery_${base_name}.res
EOF

            $exec < md_test_${base_name}.gin > md_test_${base_name}.gout
            $parser Trajectories_${base_name}.trg md_test_${base_name}.gout \
                    output_${base_name} -e E_of_t_${base_name}.dat

        done
    done
done

```

As you can see, the script is enough self-commented. Also, from line 41 to line 92 you will have already recognized the input file that we described above, with the addition that some parameters are inserted based on the variable value in the loop. Here you simply have the possibility of changing and looping over the values of time step, temperature and cell size by setting the respective "lists". You can run the script by using:

```
bash script.sh
```

or

```
nohup ./script.sh &
```

the nohup part allows the program to keep running even if you close the terminal in the computer. At the end you will find all the output files in your working directory. Type ls -l to get a list of the produced files. In the problems of this lab, you can modify the above script according to do different calculations you need.

The GULP program will generate a `.trg` trajectory file that contains the configurations generated along the MD simulation. We provide a parser in the scripts, [1] that takes the output file and trajectory of GULP and provides:

- An text file that lists for every sampled trajectory point the time, temperature, potential energy and kinetic energy. You specify the text file name with the `-e` option. This file can be easily visualized with gnuplot.
- A json-file that contains positions, velocities etc, that will be an input to the other analysis scripts we provide. The filename is the last positional input.

As a first example, `python scripts/converter_to_xcrysden.py output.json` takes the json-output you provide and writes a file that can be visualized with xcrysden, a program you have used before. Note that all possible input options to a python-script can be viewed when passing `-h`, as in `python myscript.py -h`.

[1] In order to speed up the set up and the analysis of the simulations, we provide you some useful python scripts that you can find in the scripts folder. All the scripts are intended to simplify your life during the set up of the simulations required by the different exercises. You are not forced to use them, you can analyze each calculation as you wish.

4 Silver melting

In this section, we consider the melting of silver. As appears from the GULP input file reported above, the stable crystalline form of silver at room temperature and ambient pressure is face-centered-cubic (fcc). As already mentioned, we will use an embedded atom model (EAM) potential for silver. To simulate the silver melting, we use the supercell method (bulk melting). Alternatively, one can use the surface induced melting approach, i.e. simulating the melting from a cell with a solid-vacuum interface, or one can adopt instead another approach (see Phys Rev B 49(5) 3109) that simulates the melting from a cell with a solid-liquid interface.

4.1 Hints for solving the problems

In order to solve the problems in LAB 4, you will have to run several MD calculations. In principle, you can organize directories for saving the output runs in any way you like. Usually one either creates a directory for each problem (this way one can also keep the same name for all the files), or keeps all the files in the same directory, but giving clear indicative names to each series of files belonging to the same problem. Of course the script we provided should be modified according to your choice.

4.1.1 Problem 1

Before getting started, you will need to know what temperature to look at. In order to study the melting, we need to sample below and above the silver melting point (about 960^oC, i.e. about $1235K$), so we decide to go from $1000K$ to $3000K$.[2] A $2 \times 2 \times 2$ supercell is the minimum size you need to consider, as we expect in smaller cells the fluctuations of thermodynamic properties to become very large. A good choice for the equilibration time is $1ps$.

A) In general, you should choose a `timestep` large enough in order to save CPU time, but small enough in order to minimize integration errors, that can lead to an unstable simulation with non-physical behavior. The main limitation imposed by the system here is the highest-frequency motion that must be considered: a vibrational period must be split into around 100 segments for molecular systems to satisfy the Verlet assumption that the velocities and accelerations are constant over the timestep used. A very safe timestep in this system is around 10^{-15} seconds ($= 0.001ps = 1fs$). The standard way to test the convergence of your `timestep` is to measure the energy conservation in the NVE ensemble, because in this case total energy (kinetic + potential) is expected to be constant. If the `timestep` is too big, the total energy may drift or the atoms may crash into each other. A practical hint is to use the highest temperature and the smallest supercell that you are interested in. As the temperature increases, the vibrational frequencies are higher and the vibrational periods are shorter, thus if your `timestep` is converged for the higher temperatures it will also be converged for the lower temperatures. Notice that the conserved quantity, even if there is no drift, is not

[2] Do not be worried if you get a different melting point; We are using an approximate potential, after all.

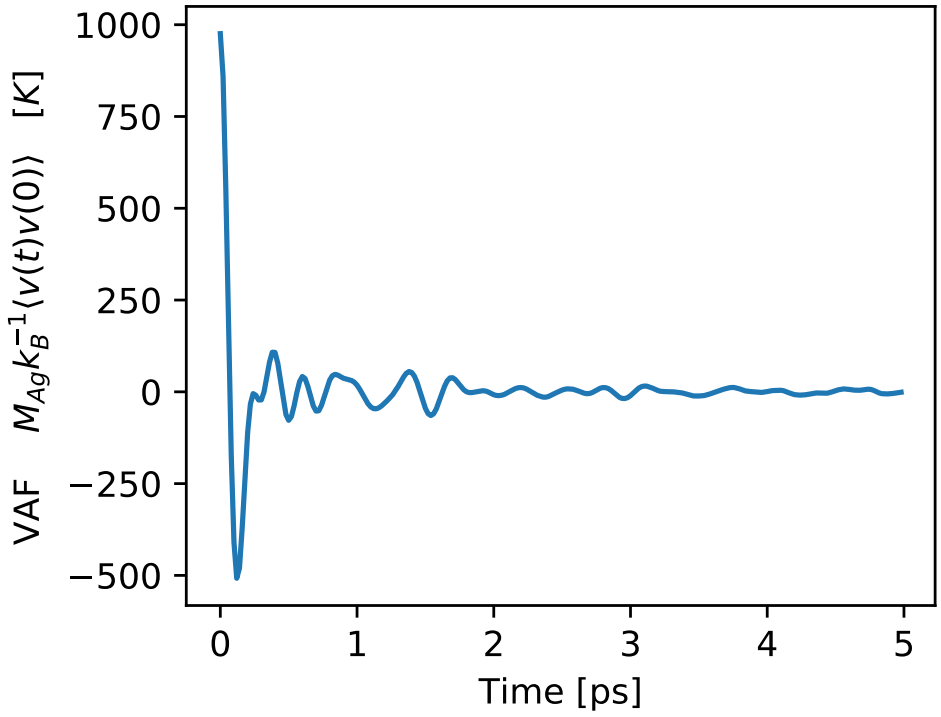

Figure 1: Velocity autocorrelation function from a silver simulation

truly constant, as there are small fluctuations that are due to the numerical integration. Thus, the amplitude of these fluctuations as a function of timestep can be used to study how the integration error scales. This convergence test should be done for a production time of $10ps$ and at least 500 samples. Remember to analyze only the production part of your simulation, the equilibration part in GULP has a velocity-rescaling algorithm to speed up the equilibration, which induces strong oscillations in the total energy.

B) Now that you have a good timestep (that you should use from now on), you are asked to check how much the temperature fluctuates around the average in an NVE simulation as a function of the supercell size. As seen above, the latter can be easily tuned in the calculation by changing the parameter "`supercell`". Fluctuations of the temperature should be around or below 5% of the target temperature, i.e. the one used in the equilibration run and set up in the input file. You should check both highest and lowest temperature. To see how the fluctuations decrease with increased system size, perform a linear fit on $\log\sigma_T^2$ against $\log N$, where N is the number of atoms in the system. Also here, $1ps$ equilibration and $10ps$ production time are fine.

C) As a last step, you should figure out how long the `production` time should be. As everything else, this quantity is also something you would ideally converge for every quantity of interest that you get get from your simulation, but this can be very cumbersome. A faster way of estimating the production time can be done if we know the effective decorrelation time of our system, that is to say the time it takes for a trajectory to lose memory. That can be estimated by looking at the velocity auto-correlation

function (VAF):

$$VAF(t) = \frac{1}{3N} \sum_{I}^{N} \sum_{\alpha=1}^{3} \langle V_{I\alpha}(t) V_{I\alpha}(0) \rangle \tag{11}$$

where $V(t)_{I\alpha}$ stands for the velocity of particle I at time t in direction α and $\langle\rangle$ denotes a time average over the trajectory. From some time t_c (called the decorrelation time) onwards, this quantity fluctuates randomly around 0. For the example in Fig. 1, this is around $2ps$. Only simulation times above t_c give us new information about the system. If you simulate $30 \times t_c$, you will have 30 independent estimates of your quantity to average, and that is usually a good enough choice. You need only one run here, with $2ps$ equilibration time and $50ps$ of production time, with the timestep and supercell size determined above.

4.1.2 Problem 2

In this problem, you will perform MD simulations in a constant temperature ensemble (NVT), to study the melting of bulk silver. In order to run in the NVT ensemble, please refer to the `ensemble nvt` part of the above commented GULP input file. You will be required to consider a significant number of different temperatures, ranging from values below the experimental melting point to temperatures significantly above it (e.g. 1000 to 3000 K). Some of these temperatures should be around the melting point. You should have a resolution in the temperature of $100K$ and use the optimized parameters from the last exercise.

A) You should use the average values of the potential energy ($\langle E_{pot} \rangle$) for each temperature (T) you calculated to obtain a plot of E versus T and check at which temperature you find a discontinuity, then, estimate the melting point of silver based on that.

B) You should estimate the melting point of silver based on the radial distribution functions of your system at different temperatures. The radial distribution function (RDF) measures the density of atoms (normalized by its average value ρ), as a function of the distance d from a reference one. It is calculated by summing the number of atoms found at a given distance in all directions. In a solid, the radial distribution function will consist of sharp peaks, whereas as a material melts these peaks will broaden and some will disappear, as you can see on fig. 2. Thus, you shall quantitatively observe the phase transition by obtaining the radial distribution functions at different temperatures.

You can use the provided script to calculate the radial distribution function as:

```
theos@theosvm:$ python /scripts/rdf.py values_string.json
```

You have several options to chose, that you can access with the `-h` option. Such options are `bin-size`, the `stepsize` that you can use to speed up the calculation by not considering every samples step, etc: for example:

```
theos@theosvm:$ python /scripts/rdf.py values_string.json --init-time
2.0 --n-bins 100 --stepsize-t 10 -o RDF.dat
```

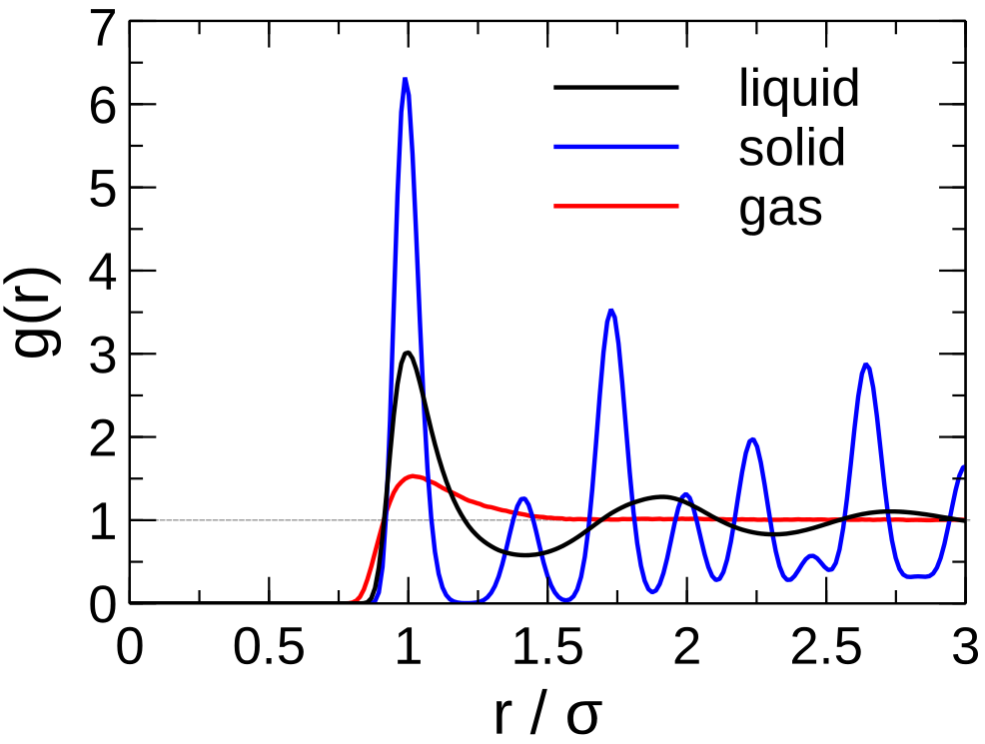

Figure 2: Radial distribution functions for different phases.

This command writes the histogram in the file (`RDF.dat` by default). As in other scripts, the initialization time is the interval of time to discard from the start of the trajectory when performing statistics. Note that the python script `rdf.py`, by default, uses only data from the production part of the GULP run, the equilibration part (see equilibration flag in the `.gin` file) is discarded by default. So, with
`initialization_time_in_picoseconds`= 0,
all data from the production part will be considered.

The script computes the RDF up to the maximum radial distance in the cell, which is half the cell size in a cubic cell. As customary, the RDF is normalized to the mean density in the system. The script also integrates the RDF, and you can use this integral to estimate the number of nearest neighbors. The distance r, RDF(r) and $\int \mathrm{RDF}(r)dr$ are written in the first, second and third column in the output file produced by the script.

The number of bins to use is a number you have to tune: too many bins will give you an histogram that is not a smooth function, whereas with too few bin you lose resolution. When you have figured out a good bin-size, plot the radial distribution functions for the different simulated temperatures, and estimate the melting temperature for silver.

C) The mean square displacements (MSD) are useful also to determine the behavior of the system of interest. With the MSD one can obtain the self diffusion-coefficient (D),

which is related to how fast the atoms are moving in the system, and can also be obtained experimentally, thus, it is a good benchmark for the simulations. The MSD is defined as:

$$\mathrm{MSD}(t) = \frac{1}{3N}\sum_{I}^{N}\sum_{\alpha=1}^{3}\langle |R_{I\alpha}(t) - R_{I\alpha}(0)|^2 \rangle \tag{12}$$

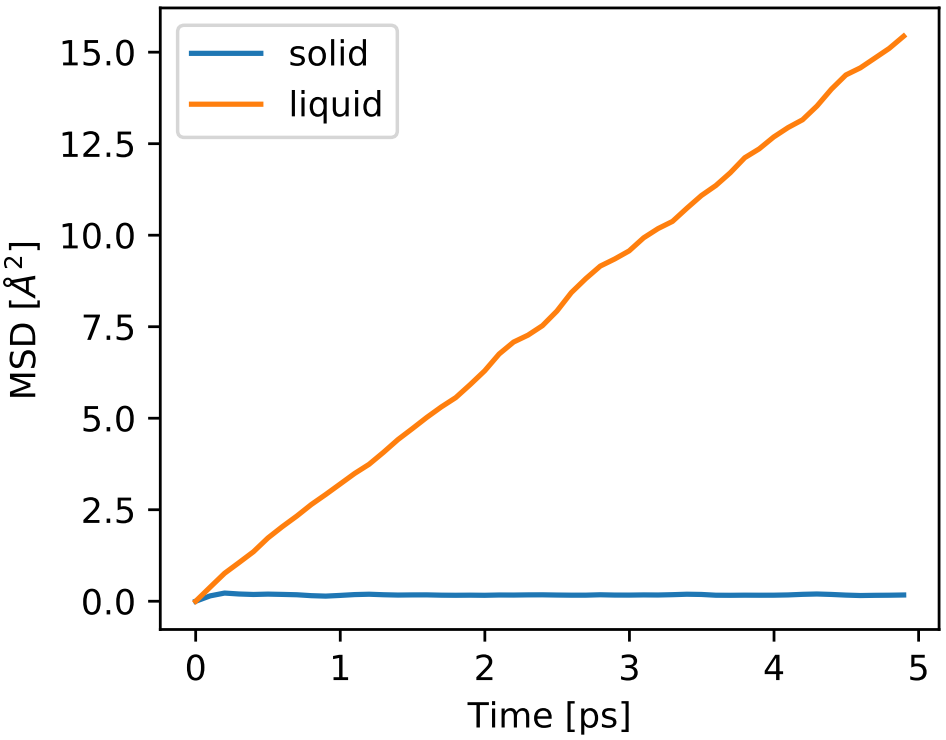

Figure 3: Mean square displacements for a solid and a liquid phase.

where $R(t)_{I\alpha}$ stands for the coordinate α of particle I at time t α and $\langle\rangle$ denotes a time average over your trajectory. From the MSD we obtain the self diffusion-coefficient, using the Einstein relation, as follows:

$$\mathrm{D} = \frac{1}{2d}\lim_{t\to\infty}\frac{d}{dt}\mathrm{MSD}(t), \tag{13}$$

where d is the dimensionality of the system, in our case it is 3, as we are dealing with a three dimensional system. One should note that the limit to infinite time is being taken, whereas the simulations are finite. Thus, a careful evaluation of the convergence as a function of the time should be performed. To calculate the self diffusion-coefficients we provide a python script `msd.py`. It can be run following the command line:

```
theos@theosvm:$ python /scripts/msd.py values_string.json
```

Here the script receives as input the name of the `.json` file produced by the python script `parser.py`. Also here, you have several options (`-h` for printing the options). The script produces an output file (specify with `-o`) that gives t, $MSD(t)$ and $\frac{d}{dt}MSD(t)$. As you learned in the course, the diffusion coefficient can be estimated from latter, but you need to give a good estimate of the slope. You can average $\frac{d}{dt}MSD(t)$ over all t where the MSD looks linear. In Fig. 3, you can safely average the slope between 1 and $5ps$, for example, in both cases.

D) A first-order phase transition is accompanied by a divergence in the specific heat capacity. This quantity can be estimated by examining the fluctuations in the total energy when simulating in the canonical (NVT) ensemble. Compute (and plot) the fixed-volume specific heat as a function of temperature, across the melting point. You may find the following relationship between the total energy fluctuations and the specific heat useful:

$$C_V = \frac{\sigma_E^2}{k_B T^2} \tag{14}$$

where σ_E^2 is the variance of the total energy. Be aware that a divergence can only be seen if you are sampling the phase transition. You might need to start a few more simulations at the melting temperature (determined before) to see the divergence.