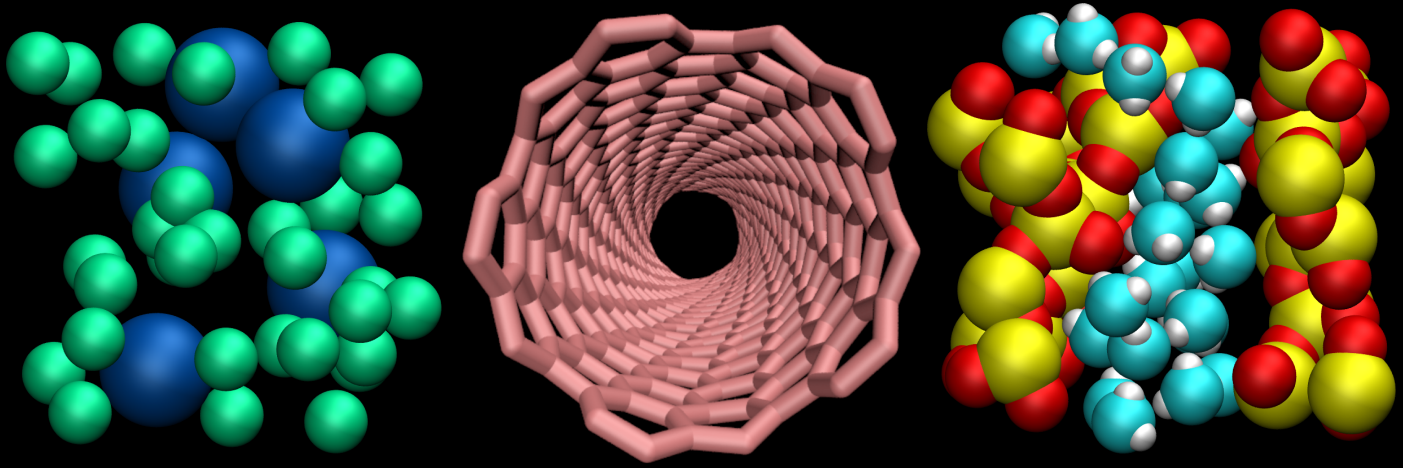
Simon Gravelle

LAMMPS tutorials



First edition

LAMMPS tutorials

First edition

Content availability

You can access the last version of *LAMMPS tutorials* from its Github repository, which comes with a webpage at *lammpstutorials.github.io*. All the simulation inputs, raw text files, and Python scripts used through the tutorials can be downloaded from a dedicated input repository.

Found any errors?

Feel free to report any issues you may encounter here, such as missing files, bugs, or typos.

Contents

Preface

LAMMPS tutorials is made of seven tutorials that are ordered by increasing difficulty. Lennard Jones fluid is meant for absolute LAMMPS and molecular dynamics beginners. The complexity of the molecular simulation is then progressively increased for Pulling on a carbon nanotube, Polymer in water, Nanosheared electrolyte, and Reactive silicon dioxide. Finally, in Water adsorption in silica and Free energy calculation, some more advanced simulation methods are used, namely grand canonical Monte Carlo simulations and a free energy method named umbrella sampling.

Required softwares

The 2 Aug 2023 version of LAMMPS is required to follow the tutorials. The other softwares listed here are optional.

LAMMPS (2 Aug 2023)

Download and install the 2 Aug 2023 version of LAMMPS by following the instructions of the LAMMPS website. Depending on your operative system (i.e. Linux, macOS, or Windows), the procedure may differ.

LAMMPS must be compiled with the following packages:

- MANYBODY
- MOLECULE
- KSPACE
- RIGID
- REAXFF
- EXTRA-DUMP

If you decide to use another LAMMPS version, certain commands may not work and LAMMPS will throw an error message.

VMD (1.9.3)

In order to visualize the simulation, the version 1.9.3 of VMD will be used. Some basic instructions for VMD are given here in the VMD tutorial. If you prefer, feel free to use an alternative visualization software like Ovito.

Python (3.11.4)

To perform post-mortem analysis of the data during the MDAnalysis tutorial, the version 2.6.1 of MDAnalysis is used together with the version 3.11.4 of Python.

In order to plot the results from the simulations, the version 3.5.2 of Matplotlib Pyplot is used in combination with lammps logfile, a library allowing one to read the *log* file produced by LAMMPS.

Text editing software

In order to write and edit LAMMPS input files, a text editor is required. Any text editor will do, such as gedit, vim, or vscode.

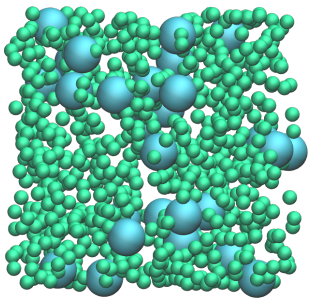
Find the input scripts

You can access all the input scripts and data files that are used in these tutorials from the inputs folder on Github. This repository also contains the full solutions to the exercises.

1

Lennard-Jones fluid

The very basics of LAMMPS through a simple example



The objective of this tutorial is to perform a simple molecular dynamics simulation of a binary fluid using LAMMPS.

The system is a Lennard-Jones fluid made of neutral particles with two different diameters in a cubic box with periodic boundary conditions. The temperature of the system is imposed using a Langevin thermostat, and some basic quantities are extracted from the system, such as the energy

This tutorial illustrates several key ingredients of molecular dynamics simulations, such as system initialization, energy minimization, integration of the equations of motion, and trajectory visualization.

1.1 The input script

In order to run a simulation using LAMMPS, one needs to write a series of commands in an input script. For clarity, this script will be divided into five categories which we are going to fill up one by one.

Create a folder, call it *my-first-input/*, and then create a blank text file in it called *input.lammps*. Copy the following lines in *input.lammps*, where a line starting with a brace (#) is a comment that is ignored by LAMMPS:

```
# PART A - ENERGY MINIMIZATION
# 1) Initialization
# 2) System definition
# 3) Simulation settings
# 4) Visualization
# 5) Run
```

These five categories are not required in every input script, and should not necessarily be in that exact order. For instance parts 3 and 4 could be inverted, or part 4 could be omitted. Note however that LAMMPS reads input file from top to bottom, therefore the *Initialization* and *System definition* categories must appear at the top of the input, and the *Run* category at the bottom.

1.1.1 System creation

In the first section of the script, called *Initialization*, let us indicate to LAMMPS the most basic information about the simulation, such as:

- the conditions at the boundaries of the box (periodic, non-periodic, ...),
- the type of atoms (uncharged single dots, spheres with angular velocities, ...).

Enter the following lines in *input.lammps*:

```
# 1) Initialization
units lj
dimension 3
atom_style atomic
pair_style lj/cut 2.5
boundary p p p
```

The first line, *units lj*, indicates that we want to use the system of unit called *LJ*, for Lennard-Jones, for which all quantities are unitless.

About Lennard-Jones (LJ) units

Lennard-Jones (LJ) units are a dimensionless system of units. LJ units are often used in molecular simulations and theoretical calculations. When using LJ units:

- energies are expressed in units of ϵ, where ϵ is the depth of the potential of the LJ interaction,
- distances are expressed in units of σ, where σ is the distance at which the particle-particle potential energy is zero,
- masses are expressed in units of the atomic mass m.

All the other quantities are normalized by a combination of ϵ, σ, and m. For instance, time is expressed in units of $\sqrt{\epsilon/m\sigma^2}$. Find on details on the LAMMPS website.

The second line, *dimension 3*, indicates that the simulation is 3D. The third line, *atom_style atomic*, that the *atomic* style will be used, therefore each atom is just a dot with a mass.

About the atom style

While we are keeping things as simple as possible in this tutorial, different *atom_style* will be used in the following tutorials. These other *atom_style* will allow us to create atoms with charges, chemical bonds, etc.

The fourth line, *pair_style lj/cut 2.5*, indicates that atoms will be interacting through a Lennard-Jones potential with a cut-off equal to $r_c = 2.5$ (unitless):

$$E_{ij}(r) = 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r}\right)^{12} - \left(\frac{\sigma_{ij}}{r}\right)^{6}\right], \text{ for } r < r_c,$$

where r is the inter-particles distance, ϵ_{ij} the depth of potential well that sets the interaction strength, and σ_{ij} the distance parameter or particle effective size. Here, the index *ij* refer to the particle types *i* and *j*.

About Lennard-Jones potential

The Lennard-Jones potential offers a simplified representation that captures the fundamental aspects of interactions among atoms. It depicts a scenario where two particles exhibit repulsion at extremely close distances, attraction at moderate distances, and no interaction at infinite separation. The repulsive part of the Lennard-Jones potential (i.e. the term $\propto r^{-12}$) is associated with the Pauli exclusion principle. The attractive part (i.e. the term in $\propto -r^{-6}$) is linked with the van der Waals forces.

The last line, *boundary p p p*, indicates that the periodic boundary conditions will be used along all three directions of space (the 3 *p* stand for *x*, *y*, and *z*, respectively).

At this point, the *input.lammps* is a LAMMPS input script that does nothing. You can run it using LAMMPS to verify that the *input* contains no mistake by running the following command in the terminal from the *my-first-input/* folder:

```
lmp -in input.lammps
```

Here *lmp* is linked to my compiled LAMMPS version. Running the previous command should return:

```
LAMMPS (2 Aug 2023 - Update 1)
Total wall time: 0:00:00
```

In case there is a mistake in the input script, for example if *atom_stile* is written instead of *atom_style*, LAMMPS gives you an explicit warning:

```
LAMMPS (2 Aug 2023 - Update 1)
ERROR: Unknown command: atom_stile  atomic (src/input.cpp:232)
Last command: atom_stile atomic
```

Let us fill the *System definition* category of the input script:

```
# 2) System definition
region simulation_box block -20 20 -20 20 -20 20
create_box 2 simulation_box
create_atoms 1 random 1500 341341 simulation_box
create_atoms 2 random 100 127569 simulation_box
```

The first line, *region simulation_box (...)*, creates a region named *simulation_box* that is a block (i.e. a rectangular cuboid) that extends from -20 to 20 (no unit) along all 3 directions of space.

The second line, *create_box 2 simulation_box*, creates a simulation box based on the region *simulation_box* with *2* types of atoms.

The third line, *create_atoms (...)* creates 1500 atoms of type 1 randomly within the region *simulation_box*. The integer *341341* is a seed that can be changed in order to create different initial conditions for the simulation. The fourth line creates 100 atoms of type 2.

If you run LAMMPS, you should see the following information in the terminal:

```
(...)
Created orthogonal box = (-20 -20 -20) to (20 20 20)
(...)
Created 1500 atoms
(...)
Created 100 atoms
(...)
```

From what is printed in the terminal, it is clear that LAMMPS correctly interpreted the commands, and first created the box with desired dimensions, then 1500 atoms, and then 100 atoms.

Let us fill the *Simulation Settings* category section of the *input* script:

```
# 3) Simulation settings
mass 1 1
mass 2 1
pair_coeff 1 1 1.0 1.0
pair_coeff 2 2 0.5 3.0
```

The two first commands, *mass (...)*, attribute a mass equal to 1 (unitless) to both atoms of type 1 and 2, respectively.alternatively, one could have written these two commands into one single line: *mass* * 1, where the star symbol means *all* the atom types of the simulation.

The third line, *pair_coeff 1 1 1.0 1.0*, sets the Lennard-Jones coefficients for the interactions between atoms of type 1, respectively the energy parameter $\epsilon_{11} = 1.0$ and the distance parameter $\sigma_{11} = 1.0$.

Similarly, the last line sets the Lennard-Jones coefficients for the interactions between atoms of type 2, $\epsilon_{22} = 0.5$, and $\sigma_{22} = 3.0$.

About cross parameters

By default, LAMMPS calculates the cross coefficients between the different atom types using geometric average: $\epsilon_{ij} = \sqrt{\epsilon_{ii}\epsilon_{jj}}$, $\sigma_{ij} = \sqrt{\sigma_{ii}\sigma_{jj}}$. In the present case, and even without specifying it explicitly, we thus have:

- $\epsilon_{ij} = \sqrt{1.0 \times 0.5} = 0.707$, and
- $\sigma_{ij} = \sqrt{1.0 \times 3.0} = 1.732$.

When necessary, cross parameters can be explicitly specifie by adding the following line to the input file: *pair_coeff 1 2 0.707 1.732*.

Note that the arithmetic rule, where $\epsilon_{ij} = \sqrt{\epsilon_{ii}\epsilon_{jj}}$, $\sigma_{ij} = (\sigma_{ii} + \sigma_{jj})/2$, is more common than the geometric rule. However, neither the geometric nor the arithmetic rule are based on rigorous arguments, so here the geometric rule will do just fine.

Due to the chosen Lennard-Jones parameters, the two types of particles are given different effective diameters, as can be seen by plotting $E_{11}(r)$, $E_{12}(r)$, and $E_{22}(r)$.

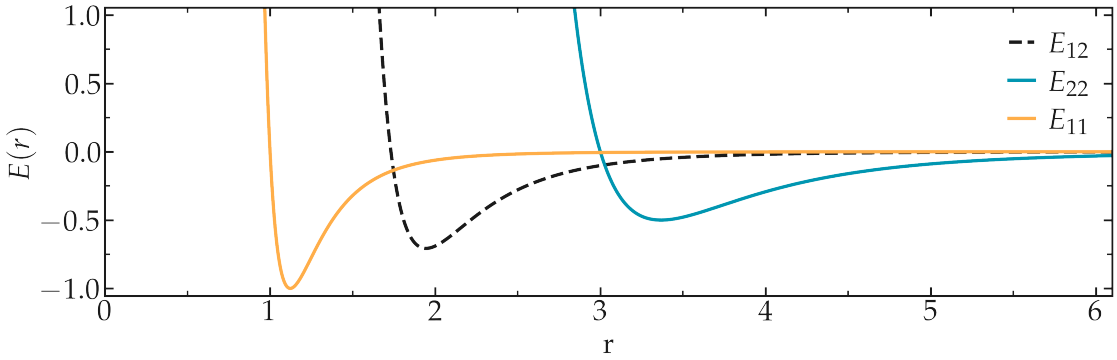


Figure: The Lennard-Jones potential $E_{ij}(r)$, where $i, j = 1$ or 2.

1.1.2 Energy minimization

The system is now fully parametrized. Let us fill the two last remaining sections by adding the following lines to *input.lammps*:

```
# 4) Visualization
thermo 10
thermo_style custom step temp pe ke etotal press

# 5) Run
minimize 1.0e-4 1.0e-6 1000 10000
```

The *thermo* command asks LAMMPS to print thermodynamic information (e.g. temperature, energy) in the terminal every given number of steps, here 10 steps. The *thermo_style custom* requires LAMMPS to print the system temperature (*temp*), potential energy (*pe*), kinetic energy (*ke*), total energy (*etotal*), and pressure (*press*). Finally, the *minimize* line asks LAMMPS to perform an energy minimization of the system.

About energy minimization

An energy minimization procedure consists in adjusting the coordinates of the atoms that are too close to each other until one of the stopping criteria is reached. By default, LAMMPS uses the conjugate gradient (CG) algorithm. Here, there are four stopping criteria:

- The change in energy between two iterations is less than 1.0e-4,
- The maximum force between two atoms in the system is lower than 1.0e-6,
- The maximum number of iterations is 1000,
- The maximum number of times the force and the energy have been evaluated is 10000.

Now running the simulation, we can see how the thermodynamic variables evolve with time:

```
Step  Temp  PotEng         KinEng    TotEng         Press
0     0     78840982       0         78840982       7884122
10    0     169.90532      0         169.90532      17.187291
20    0    -0.22335386     0        -0.22335386    -0.0034892297
30    0    -0.31178296     0        -0.31178296    -0.0027290466
40    0    -0.38135002     0        -0.38135002    -0.0016419218
50    0    -0.42686621     0        -0.42686621    -0.0015219081
60    0    -0.46153953     0        -0.46153953    -0.0010659992
70    0    -0.48581568     0        -0.48581568    -0.0014849169
80    0    -0.51799572     0        -0.51799572    -0.0012995545
(...)
```

These lines give us information about the progresses of the energy minimization. First, at the start of the simulation (Step 0), the energy in the system is huge: 78840982 (unitless). This was expected because the atoms have been created at random positions within the simulation box, and some of them are probably overlapping, resulting in a large initial energy which is the consequence of the repulsive part of the Lennard-Jones interaction potential. As the energy minimization progresses, the energy rapidly decreases and reaches a negative value, indicating that the atoms have been displaced at reasonable distances from each others.

Other useful information has been printed in the terminal, for example, LAMMPS tells us that the first of the four criteria to be satisfied was the energy:

```
Minimization stats:
Stopping criterion = energy tolerance
```

1.1.3 Molecular dynamics

The system is now ready. Let us continue filling up the input script and adding commands in order to perform an actual molecular dynamics simulation that will start from the final state of the energy minimization.

Background Information – What is molecular dynamics?

Molecular dynamics (MD) is based on the numerical solution of the Newtonian equations of motion for every atom i,

$$\sum_{j \neq i} \boldsymbol{F}_{ji} = m_i \times \boldsymbol{a}_i,$$

In the same input script, after the *minimization* command, add the following lines:

```
# PART B - MOLECULAR DYNAMICS
# 4) Visualization
thermo 50

# 5) Run
fix mynve all nve
fix mylgv all langevin 1.0 1.0 0.1 1530917
timestep 0.005
run 10000
```

Since LAMMPS reads the input from top to bottom, these lines will be executed after the energy minimization. There is no need to re-initialize the system re-define it, or re-specify the settings. The *thermo* command is called a second time within the same input, so the previously entered value of *10* will be replaced by the value of *50* as soon as *PART B* starts.

In the run section, the *fix nve* is used to update the positions and the velocities of the atoms in the group *all*. Therefore, *fix nve* contains the time integrator and is the most important command here.

What is a fix?

In LAMMPS, a *fix* corresponds to an operation that is applied to the system.

The second fix applies a Langevin thermostat to the atoms of group *all*, with a desired temperature of 1 and a *damping* parameter of 0.1. The number *1530917* is a seed, you can change it to perform statistically independent simulations. Finally we choose the *timestep* and we ask LAMMPS to run for 10000 steps. After running the simulation, you should see the following information in the terminal:

```
Step    Temp          PotEng         KinEng        TotEng         Press
388     0             -0.95476642    0            -0.95476642   -0.000304834
400     0.68476875    -0.90831467    1.0265112     0.11819648    0.023794293
500     0.97168188    -0.56803405    1.4566119     0.88857783    0.02383215
600     1.0364167     -0.44295618    1.5536534     1.1106972     0.027985679
700     1.010934      -0.39601767    1.5154533     1.1194356     0.023064983
800     0.98641731    -0.37866057    1.4787012     1.1000406     0.023131153
900     1.0074571     -0.34951264    1.5102412     1.1607285     0.023520785
(...)
```

The second column shows that the temperature *Temp* starts from 0, but rapidly reaches the requested value and stabilize itself near $T = 1$.

From what has been printed in the *log* file, one can plot the potential energy (p_e) and the kinetic energy (k_e) of the system over time (see the figure below).

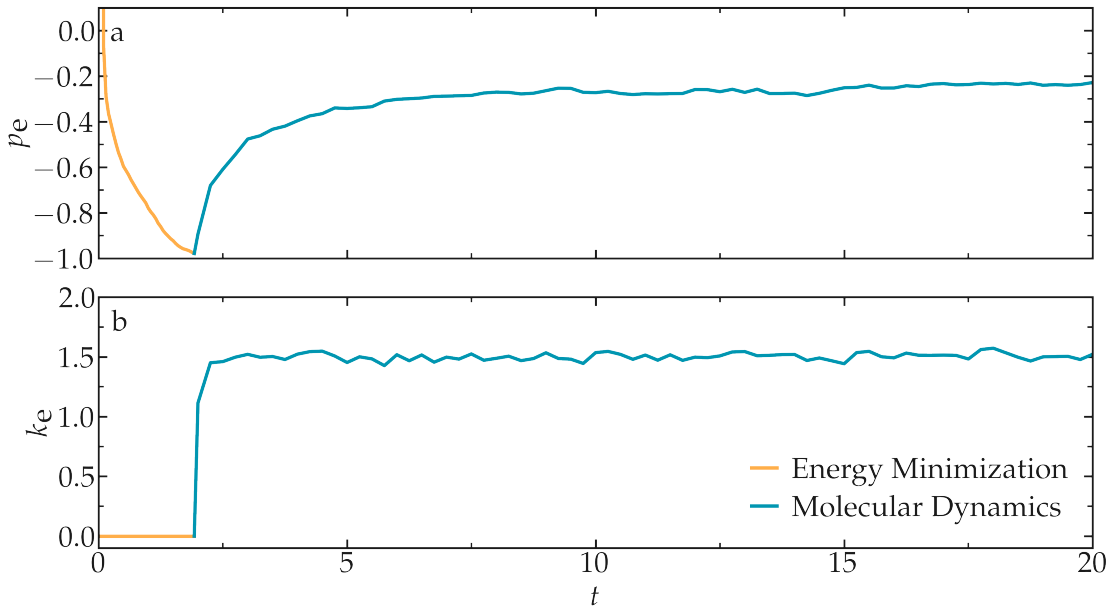


Figure: a) The potential energy (p_e) rapidly decreases during energy minimization (orange). Then, after the molecular dynamics starts, p_e increases until it reaches a plateau value of about -0.25 (blue). b) The kinetic energy (k_e) is equal to zero during energy minimization, then increases during molecular dynamics until it reaches a plateau value of about 1.5.

1.1.4 Trajectory visualization

The simulation is running well, but we would like to visualize the trajectories of the atoms. To do so, we need to dump the positions of the atoms in a file at a regular interval.

Add the following command to the *input.lammps* file, in the *visualization* section of PART B:

```
dump mydmp all atom 100 dump.lammpstrj
```

Run the *input.lammps* using LAMMPS again. A file named *dump.lammpstrj* must appear within *my-first-input/*. A *.lammpstrj* file can be opened using VMD. With Ubuntu/Linux, you can simply execute in the terminal:

```
vmd dump.lammpstrj
```

Otherwise, you can open VMD and import the *dump.lammpstrj* file manually using *File -¿ New molecule*.

By default, you should see a cloud of lines, but you can improve the representation (see this VMD tutorial for basic instructions).

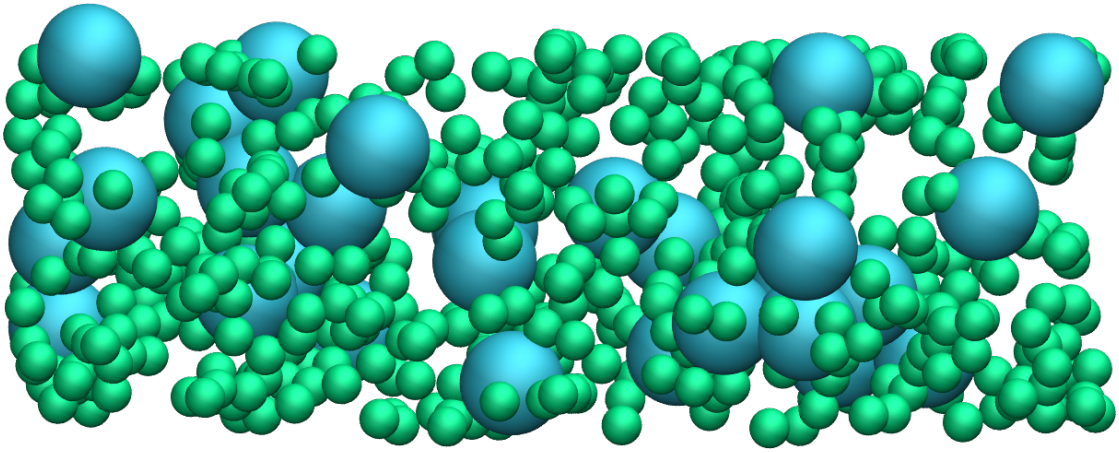


Figure: View of a slice of the system using VMD, with both types of atoms represented as spheres. See the corresponding video.

1.2 Improving the script

Let us improve the input script and perform slightly more advanced operations, such as imposing a specific initial positions to the atoms, and restarting the simulation from a previously saved configuration.

1.2.1 Control the initial atom positions

Let us create the atoms of type 1 and 2 in two separate regions, respectively, instead of creating them both randomly within the entire space as we did previously.

Create a new folder next to *my-first-input/*, and call it *improved-input/*. Then, create a new input and call it *input.min.lammps*.

Similarly to what has been done previously, copy the following lines into *input.min.lammps*:

```
# 1) Initialization
units lj
dimension 3
atom_style atomic
pair_style lj/cut 2.5
boundary p p p
```

Let us create three separate regions: A cubic region for the simulation box and two additional regions for placing the atoms:

```
# 2) System definition
region simulation_box block -20 20 -20 20 -20 20
create_box 2 simulation_box
region region_cylinder_in cylinder z 0 0 10 INF INF side in
region region_cylinder_out cylinder z 0 0 10 INF INF side out
create_atoms 1 random 1000 341341 region_cylinder_out
create_atoms 2 random 150 127569 region_cylinder_in
```

The *side in* and *side out* keywords are used to define regions that are respectively inside and outside of the cylinder of radius 10. Then, copy similar lines as previously into *input.min.lammps*:

```
# 3) Simulation settings
mass 1 1
mass 2 1
pair_coeff 1 1 1.0 1.0
pair_coeff 2 2 0.5 3.0

# 4) Visualization
thermo 10
thermo_style custom step temp pe ke etotal press
dump mydmp all atom 10 dump.min.lammpstrj

# 5) Run
minimize 1.0e-4 1.0e-6 1000 10000
write_data minimized_coordinate.data
```

The main novelty, with respect to the previous input script, is the *write_data* command. This command is used to print the final state of the simulation in a file named *minimized_coordinate.data*. Note that the *write_data* command is placed after the *minimize* command. This *.data* file will be used later to restart the simulation from the final state of the energy minimisation step.

Run the *input.min.lammps* script using LAMMPS. A new dump file named *dump.min.lammpstrj* will appear in the folder, allowing you to visualize the atoms trajectories during minimization. In addition, a file named *minimized_coordinate.data* will be created.

If you open *minimized_coordinate.data*, you will see that it contains all the information necessary to restart the simulation, such as the number of atoms and the size of the box. The *.data* file even contains the atoms *masses* and *pair_coeffs*:

```
1150 atoms
2 atom types

-20 20 xlo xhi
-20 20 ylo yhi
-20 20 zlo zhi

Masses

1 1
2 1

Pair Coeffs # lj/cut

1 1 1
2 0.5 3
(...)
```

The *minimized_coordinate.data* file also contains the final positions and velocities of all the atoms:

```
(...)
Atoms # atomic

970 1 4.4615279184230525 -19.88248310680258 -19.497251754277872 0 0 0
798 1 1.0773937287460968 -17.57843015813612 -19.353475858951473 0 0 0
21 1 -17.542385434367777 -16.647460269156497 -18.93914807895693 0 0 0
108 1 -15.96241088290946 -15.956274144833264 -19.016419910024062 0 0 0
351 1 0.08197850837343444 -16.852380573900156 -19.28249747472579 0 0 0
402 1 -5.270160783673711 -15.592291204068946 -19.6382667867645 0 0 0
(...)
```

The columns of the *Atoms* section correspond (from left to right) to the atom indexes (from 1 to the total number of atoms, 1150), the atom types (1 or 2 here), the atoms positions x, y, z. The last three columns are image flags that keep track of which atoms crossed the periodic boundary.

1.2.2 Restarting from a saved configuration

Let us create a new input file and start a molecular dynamics simulation directly from the previously saved configuration. Within *improved-input/*, create a new file named *input.md.lammps* and copy the same lines as previously:

```
# 1) Initialization
units lj
dimension 3
atom_style atomic
pair_style lj/cut 2.5
boundary p p p
```

Now, instead of creating a new region and adding atoms to it, we can simply add the following command:

```
# 2) System definition
read_data minimized_coordinate.data
```

By visualizing the previously generated dump.min.lammpstrj file, you may have noticed that some atoms have moved from one region to the other during minimisation. In order to start the simulation from a clean state, with only atoms of type 2 within the cylinder and atoms of type 1 outside the cylinder, let us delete the misplaced atoms by adding the following commands to *input.md.lammps*:

```
read_data minimized_coordinate.data
region region_cylinder_in cylinder z 0 0 10 INF INF side in
region region_cylinder_out cylinder z 0 0 10 INF INF side out
group group_type_1 type 1
group group_type_2 type 2
group group_region_in region region_cylinder_in
group group_region_out region region_cylinder_out
group group_type_1_in intersect group_type_1 group_region_in
group group_type_2_out intersect group_type_2 group_region_out
delete_atoms group group_type_1_in
delete_atoms group group_type_2_out
```

The two first *region* commands recreate the previously defined regions, which is necessary since regions are not saved by the *write_data* command.

The first two *group* commands create atom groups based on their types. The next two *group* commands create atom groups based on their positions at the beginning of the simulation, i.e. when the commands are being read by LAMMPS. The last two *group* commands create atom groups based on intersection between the previously defined groups.

Finally, the two *delete_atoms* commands delete the atoms of type 1 that are located within the cylinder, as well as the atoms of type 2 that are located outside the cylinder, respectively.

When you run the *input.md.lammps* input using LAMMPS, you can see in the *log* file how many atoms are in each group, and how many atoms have been deleted:

```
1000 atoms in group group_type_1
150 atoms in group group_type_2
149 atoms in group group_region_in
1001 atoms in group group_region_out
0 atoms in group group_type_1_in
1 atoms in group group_type_2_out
Deleted 0 atoms, new total = 1150
Deleted 1 atoms, new total = 1149
```

Add the following lines to *input.md.lammps*. Note the absence of *Simulation settings* section, because the settings are taken from the *.data* file.

```
# 4) Visualization
thermo 1000
dump mydmp all atom 1000 dump.md.lammpstrj
```

Let us extract the number of atoms of each type inside the cylinder as a function of time, by adding the following commands to *input.md.lammps*:

```
variable number_type1_in equal count(group_type_1,region_cylinder_in)
variable number_type2_in equal count(group_type_2,region_cylinder_in)
fix myat1 all ave/time 10 200 2000 v_number_type1_in &
    file output-population1vstime.dat
fix myat2 all ave/time 10 200 2000 v_number_type2_in &
    file output-population2vstime.dat
```

The 2 *variables* are used to count the number of atoms of a specific group in the *region_cylinder_in* region.

The two *fix ave/time* are calling the previously defined variables and are printing their values into text files. By using *10 200 2000*, variables are evaluated every 10 steps, averaged 200 times, and printed in the *.dat* files every 2000 steps.

Let us also extract the coordination number per atom between atoms of type 1 and 2, i.e. the average number of atoms of type 2 in the vicinity of the atoms of type 1. This coordination number will be used as an indicator of the degree of mixing of our binary mixture. Add the folloxing lines into *input.md.lammps*:

```
compute coor12 group_type_1 coord/atom cutoff 2.0 group group_type_2
compute sumcoor12 all reduce ave c_coor12
fix myat3 all ave/time 10 200 2000 &
    c_sumcoor12 file coordinationnumber12.dat
```

The *compute ave* is used to average the per atom coordination number that is calculated by the *coord/atom* compute. This averaging is necessary as *coord/atom* returns an array where each value corresponds to a certain couple of atom i-j. Such array can't be printed by *fix ave/time*. Finally, let us complete the script by adding the following lines to *input.md.lammps*:

```
# 5) Run
velocity all create 1.0 4928459 mom yes rot yes dist gaussian
fix mynve all nve
fix mylgv all langevin 1.0 1.0 0.1 1530917 zero yes
timestep 0.005
run 300000
write_data mixed.data
```

There are a few differences with the previous simulation. First, the *velocity create* command attributes an initial velocity to every atom. The initial velocity is chosen so that the average initial temperature is equal to 1 (unitless). The additional keywords ensure that no linear momentum (*mom yes*) and no angular momentum (*rot yes*) are given to the system, and that the generated velocities are distributed as a Gaussian. Another improvement is the *zero yes* keyword in the Langevin thermostat, that ensures that the total random force is equal to zero.

Run *input.md.lammps* using LAMMPS and vizualize the trajectory using VMD:

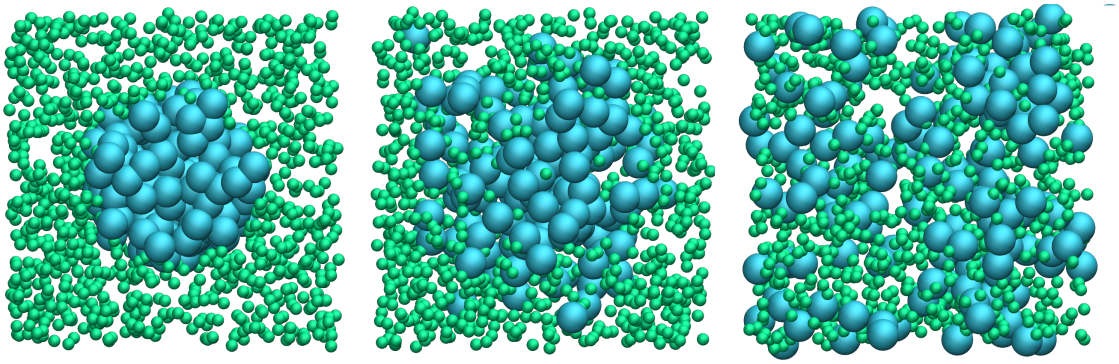


Figure: Evolution of the atom populations during mixing.

After running *input.md.lammps* using LAMMPS, you can observe the number of atoms in each region from the generated data files, as well as the evolution of the coordination number due to mixing:

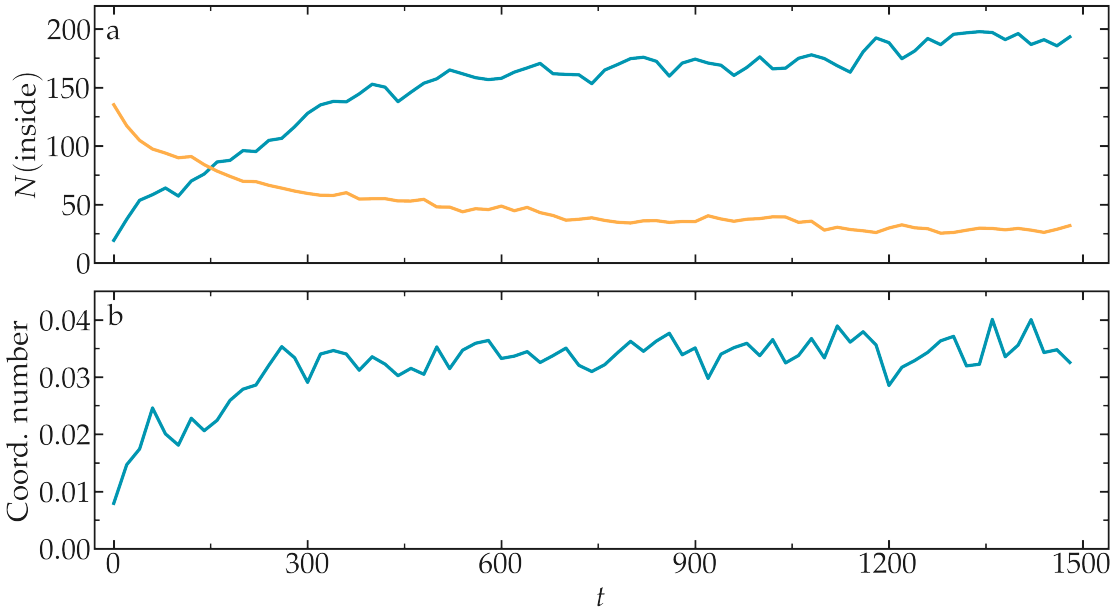


Figure: Evolution of the number of atoms within the *region_cylinder_in* region as a function of time (a), and evolution of the coordination number (b).

You can access all the input scripts and data files that are used in these tutorials from the inputs folder on Github. This repository also contains the full solutions to the exercises.

1.3 Going further with exercises

Each exercise comes with a proposed solution, see Solutions to the exercises.

1.3.1 Solve Lost atoms error

For this exercise, the following input script is provided:

```
units lj
dimension 3
atom_style atomic
pair_style lj/cut 2.5
boundary p p p

region simulation_box block -20 20 -20 20 -20 20
create_box 1 simulation_box
create_atoms 1 random 1000 341841 simulation_box

mass 1 1
pair_coeff 1 1 1.0 1.0

dump mydmp all atom 100 dump.lammpstrj
thermo 100
thermo_style custom step temp pe ke etotal press

fix mynve all nve
fix mylgv all langevin 1.0 1.0 0.1 1530917
timestep 0.005

run 10000
```

As it is, this input returns one of the most common error that you will encounter using LAMMPS:

```
ERROR: Lost atoms: original 1000 current 984
```

The goal of this exercise is to fix the *Lost atoms* error without using any other command than the ones already present. You can only play with the values of the parameters and/or replicate every command at many times as needed.

> **Note**
>
> This script is failing because particles are created randomly in space, some of them are likely overlapping, and no energy minimization is performed prior to start the molecular dynamics simulation.

1.3.2 Create a demixed dense phase

Starting from one of the *input* created in this tutorial, fine tune the parameters such as particle numbers and interaction to create a simulation with the following properties:

- the density in particles must be high,
- both particles of type 1 and 2 must have the same size,
- particles of type 1 and 2 must demix.

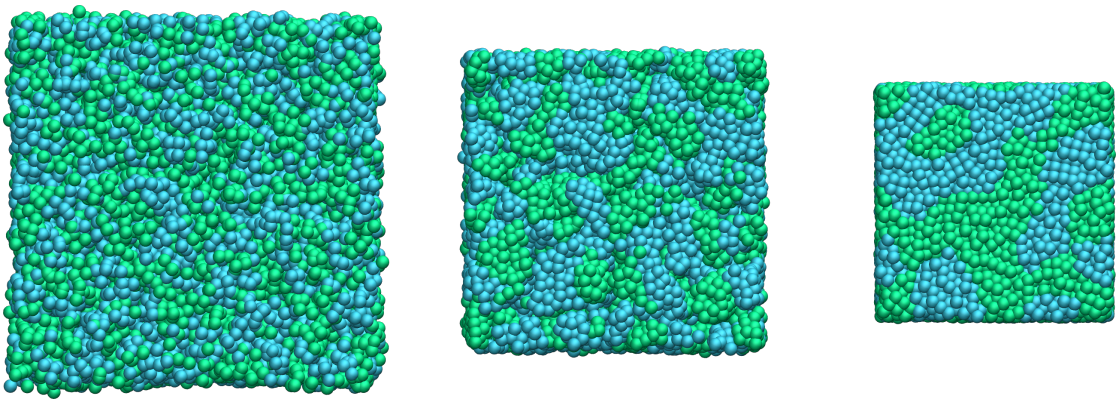


Figure: Snapshots taken at different times showing the particles of type 1 and type 2 progressively demixing and forming large demixed areas.

Hint

An easy way to create a dense phase is to allow the box dimensions to relax until the vacuum disapears. You can do that by replacing the *fix nve* by *fix nph*.

1.3.3 From atoms to molecules

Add a bond between particles of *type 2* to create dumbbell molecules instead of single paticles.

Similarly, can you create a small polymer, i.e. a long chain of particles linked by bonds and angles?

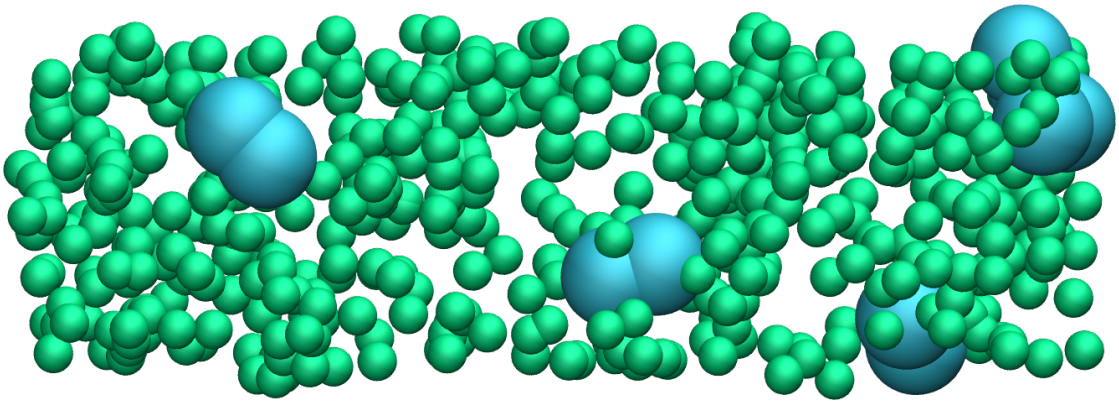


Figure: Dumbbell molecules made of 2 large spheres mixed with smaller particles (small spheres). See the corresponding video.

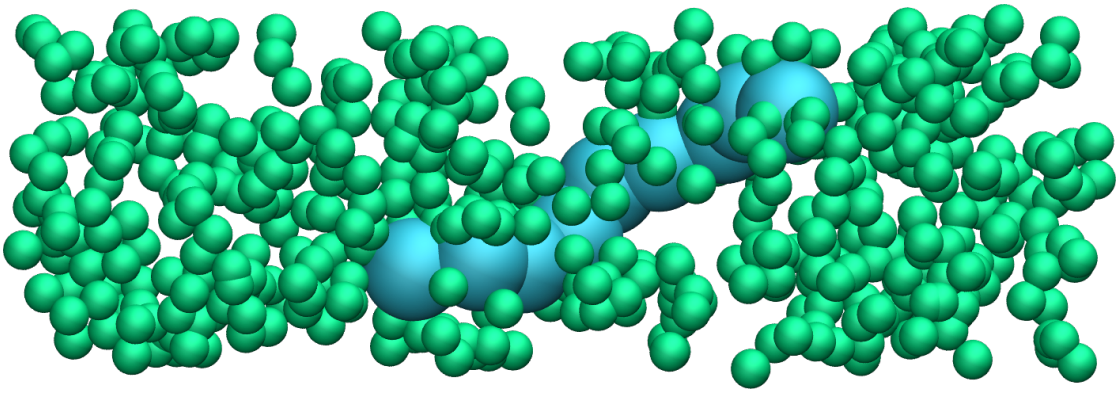


Figure: A single small polymer molecule made of 9 large spheres mixed with smaller particles. See the corresponding video.

Hints

Use a *molecule template* to easily insert as many atoms connected by bonds (i.e. molecules) as you want. A molecule template typically begins as follow:

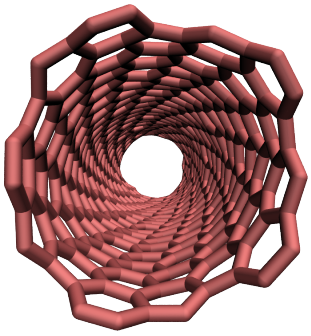
```
2 atoms
1 bonds

Coords

1 0.5 0 0
2 -0.5 0 0

(...)
```

A bond section also needs to be added.

2

Pulling on a carbon nanotube

Stretching a carbon nanotube until it breaks



The objective of this tutorial is to impose the deformation of a carbon nanotube (CNT) using LAMMPS.

In this tutorial, a small carbon nanotube (CNT) is simulated within an empty box using LAMMPS. An external forcing is imposed to the CNT, and its deformation is measured with time.

The difference between classical and reactive force fields is illustrated through this tutorial. With a classical force field, the bonds between atoms are unbreakable. With the reactive force field (named AIREBO), the breaking of the chemical bonds is possible when the imposed deformation is strong enough.

If you are completely new to LAMMPS, I recommend you to follow this tutorial on a simple Lennard Jones fluid first.

2.1 Unbreakable bonds

With most classical molecular dynamics force fields, the chemical bonds between the atoms are set at the start of the simulation. Regardless of the forces applied on the atoms during the simulations, the bonds remain intact. The bonds between neighbor atoms typically consist of springs with given equilibrium distances r_0 and constant k_b:

$$U_b = k_b \left(r - r_0\right)^2 .$$

Additionally, angular and dihedral constraints are usually applied to maintain the relative orientations of neighbor atoms.

2.1.1 Create topology with VMD

When the system has a complex topology, like is the case of a CNT, it is better to use an external preprocessing tool to create it. Many preprocessing tools exist, see a non-exhaustive list on the LAMMPS website. Here, VMD is used. Alternatively, you can skip this part of the tutorial by downloading the CNT topology I did create by clicking here, and continue with the tutorial.

Open VMD, go to Extensions, Modeling, and then Nanotube Builder. A window named Carbon Nanostructures opens up, allowing us to choose between generating a sheet or a nanotube, made either of graphene or of Boron Nitride (BN). For this tutorial, let us generate a carbon nanotube. Keep all default values, and click on Generate Nanotube.

At this point, this is not a molecular dynamics simulation, but a cloud of unconnected dots. In the VMD terminal, set the box dimensions by typing the following commands in the VMD terminal:

```
molinfo top set a 80
molinfo top set b 80
molinfo top set c 80
```

The values of 80 in each direction have been chosen so that the box is much larger than the carbon nanotube.

In order to generate the initial LAMMPS data file, let us use Topotool: to generate the LAMMPS data file, enter the following command in the VMD terminal:

```
topo writelammpsdata cnt_molecular.data molecular
```

Here *molecular* refers to the LAMMPS *atom_style*, and *cnt_molecular.data* is the name of the file.

About TopoTools

More details about Topotool can be found on the personal page of Axel Kohlmeyer. In short, Topotools deduces the location of bonds, angles, dihedrals, and impropers from the respective positions of the atoms, and generates a *.data* file that can be read by LAMMPS.

The parameters of the constraints (bond length, dihedral coefficients, etc.) will be given later. A new file named *cnt_molecular.data* has been created, it starts like that:

```
700 atoms
1035 bonds
2040 angles
4030 dihedrals
670 impropers
1 atom types
1 bond types
1 angle types
1 dihedral types
1 improper types
-40.000000 40.000000  xlo xhi
-40.000000 40.000000  ylo yhi
-12.130411 67.869589  zlo zhi
(...)
```

The *cnt_molecular.data* file contains information about the positions of the carbons atoms, as well as the identity of the atoms that are linked by *bonds*, *angles*, *dihedrals*, and *impropers* constraints.

Save the *cnt_molecular.data* file in a folder named *unbreakable-bonds/*.

2.1.2 The LAMMPS input

Create a new text file within *unbreakable-bonds/* and name it *input.lammps*. Copy the following lines in it:

```
variable T equal 300

units real
atom_style molecular
boundary f f f
pair_style lj/cut 14

bond_style harmonic
angle_style harmonic
dihedral_style opls
improper_style harmonic

special_bonds lj 0.0 0.0 0.5

read_data cnt_molecular.data
```

The chosen unit system is *real* (therefore distances are in Angstrom, time in femtosecond), the *atom_style* is molecular (therefore atoms are dots that can be bonded with each other), and the boundary conditions are fixed. The boundary conditions do not really matter here, as the box boundaries were placed far from the CNT.

Just like in Lennard Jones fluid, the pair style is *lj/cut* (i.e. a Lennard-Jones potential with a short range cutoff) with parameter 14, which means that only the atoms closer than 14 Angstroms from each others interact through a Lennard-Jones potential.

The *bond_style*, *angle_style*, *dihedral_style*, and *improper_style* commands specify the different potentials used to restrain the relative positions of the atoms. For more details about the potentials

used here, you can have a look at the LAMMPS website, see for example the page of the OPLS dihedral style.

The last command, *read_data*, imports the *cnt_molecular.data* file previously generated with VMD, which contains the information about the box size, atoms positions, etc.

About interaction between neighbors atoms

We need to specify the parameters of both bonded and non-bonded potentials. Create a new text file in the *unbreakable-bonds/* folder and name it *parm.lammps*. Copy the following lines in it:

```
pair_coeff 1 1 0.066047 3.4
bond_coeff 1 469 1.4
angle_coeff 1 63 120
dihedral_coeff 1 0 7.25 0 0
improper_coeff 1 5 180
```

The *pair_coeff* command sets the LJ parameters ϵ and σ for the only type of atom of the simulation: carbon atom of type 1. The *bond_coeff* provides the equilibrium distance r_0 as well as the spring constant k_b for the harmonic potential imposed between two neighboring carbon atoms, where the potential is $U_b = k_b(r - r_0)^2$. The *angle_coeff* gives the equilibrium angle $theta_0$ and constant for the potential between three neighbors atoms : $U_\theta = k_\theta(\theta - \theta_0)^2$. The *dihedral_coeff* and *improper_coeff* give the potential for the constraints between 4 atoms.

The file *parm.lammps* is included in the simulation by adding the following line to the *input.lammps* file:

```
include parm.lammps
```

2.1.3 Prepare initial state

Before starting the molecular dynamics simulation, let us make sure that we start from a clean initial state by recentering the CNT at the origin (0, 0, 0). In addition, let us make sure that the box boundaries are symmetric with respect to (0, 0, 0), which is not initially the case, as seen in *cnt_molecular.data*:

```
-40.000000 40.000000  xlo xhi
-40.000000 40.000000  ylo yhi
-12.130411 67.869589  zlo zhi
```

Let us recenter the CNT by adding the following lines to *input.lammps*:

```
group carbon_atoms type 1
variable carbon_xcm equal -1*xcm(carbon_atoms,x)
variable carbon_ycm equal -1*xcm(carbon_atoms,y)
variable carbon_zcm equal -1*xcm(carbon_atoms,z)
displace_atoms carbon_atoms &
    move ${carbon_xcm} ${carbon_ycm} ${carbon_zcm}
```

The first command includes all the atoms of type 1 (i.e. all the atoms here) in a group named *carbon_atoms*. The 3 variables, *carbon_xcm*, *carbon_ycm*, and *carbon_zcm* are used to measure the current position of the group *carbon_atoms* along all 3 directions, respectively. Then, the *displace_atoms* command move the group *carbon_atoms*, ensuring that its center of mass is located at the origin (0, 0, 0).

Let us also change the box boundaries by adding the following line to *input.lammps*:

```
change_box all x final -40 40 y final -40 40 z final -40 40
```

Note

Such cleaner and more symmetrical initial state can simplify future data analysis, but won't make any difference to the molecular dynamics.

A displacement will be imposed to the edges of the CNT. To do so, let us isolate the atoms from the two edges and place them into groups named *rtop* and *rbot*, respectively. Add the following lines to *input.lammps*:

```
variable zmax equal bound(carbon_atoms,zmax)-0.5
variable zmin equal bound(carbon_atoms,zmin)+0.5
region rtop block INF INF INF INF ${zmax} INF
region rbot block INF INF INF INF INF ${zmin}
region rmid block INF INF INF INF ${zmin} ${zmax}
```

The variable $z_{\max}$ corresponds to the coordinate of the last atoms along z minus 0.5 Angstroms, and $z_{\min}$ to the coordinate of the first atoms along z plus 0.5 Angstroms. Then, 3 regions are defined, and correspond respectively to: $z < z_{\min}$, (bottom) $z_{\min} > z > z_{\max}$ (middle), and $z > z_{\max}$ (top).

Finally, let us define 3 groups of atoms corresponding to the atoms located in each of the 3 regions, respectively, by adding to *input.lammps*:

```
group carbon_top region rtop
group carbon_bot region rbot
group carbon_mid region rmid
```

The atoms of the edges as selected within the *carbon_top* and *carbon_bot* groups can be represented with a different color.

When running a simulation, the number of atoms in each group is printed in the terminal (and in the log.lammps file). Always make sure that the number of atoms in each group corresponds to what is expected, just like here:

```
10 atoms in group carbon_top
10 atoms in group carbon_bot
680 atoms in group carbon_mid
```

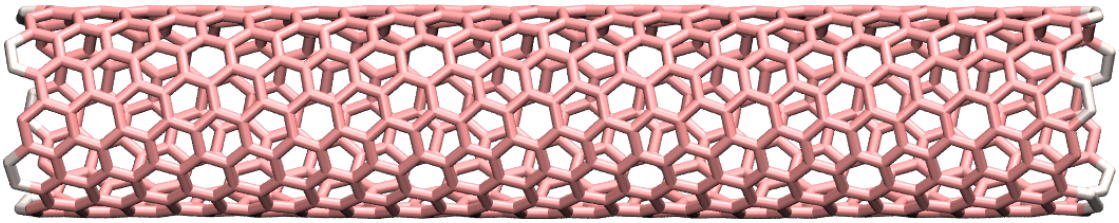


Figure: CNT with atoms from the *carbon_top* and the *carbon_bot* groups being represented with a different color.

Finally, let us randomly delete some of the carbon atoms. In order to avoid deleting atoms that are too close from the edges, let us define a new region name *rdel* that starts 2 $\mathring{A}$ from the CNT edges.

```
variable zmax_del equal ${zmax}-2
variable zmin_del equal ${zmin}+2
region rdel block INF INF INF INF ${zmin_del} ${zmax_del}
group rdel region rdel
delete_atoms random fraction 0.02 no rdel NULL 482793 bond yes
```

The *delete_atoms* command randomly deletes 2 % of the atoms from the *rdel* group (i.e. about 10 atoms).

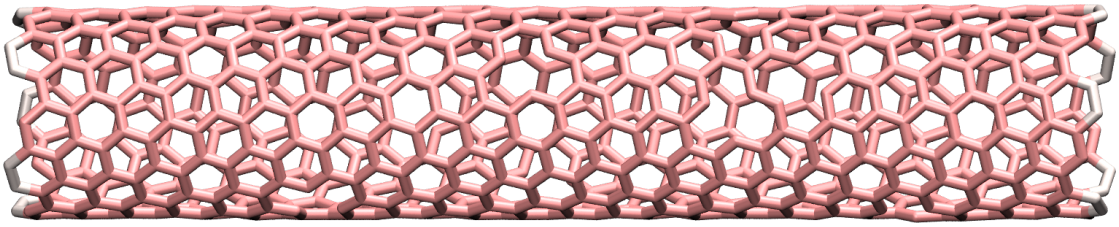


Figure: CNT with *10* randomly deleted atoms.

2.1.4 The molecular dynamics

Let us specify the thermalization and the dynamics of the system. Add the following lines to *input.lammps*:

```
reset_atoms id sort yes
velocity carbon_mid create ${T} 48455 mom yes rot yes
fix mynve all nve
compute Tmid carbon_mid temp
fix myber carbon_mid temp/berendsen ${T} ${T} 100
fix_modify myber temp Tmid
```

Re-setting the atoms ids is necessary before using the *velocity* command, this is done by the *reset_atoms* command.

The *velocity* command gives initial velocities to the atoms of the middle group *carbon_mid*, ensuring an initial temperature of 300 K for these atoms with no overall translational momentum, *mom yes*, nor rotational momentum, *rot yes*.

The *fix nve* is applied to all atoms so that all atom positions are recalculated at every step, and a *Berendsen* thermostat is applied to the atoms of the group *carbon_mid* only. The *fix_modify myber* ensures that the *fix Berendsen* uses the temperature of the group *carbon_mid* as an input, instead of the temperature of whole system. This is necessary to make sure that the frozen edges won't bias the temperature. Note that the atoms of the edges do not need a thermostat because their motion will be restrained, see below.

Deal with semi-frozen system

Always be careful when part of a system is frozen, as is the case here. When some atoms are frozen, the total temperature of the system is effectively lower than the applied temperature because the frozen atoms have no thermal motion (their temperature is therefore 0 K).

To restrain the motion of the atoms at the edges, let us add the following commands to *input.lammps*:

```
fix mysf1 carbon_top setforce 0 0 0
fix mysf2 carbon_bot setforce 0 0 0
velocity carbon_top set 0 0 0
velocity carbon_bot set 0 0 0
```

The two *setforce* commands cancel the forces applied on the atoms of the two edges, respectively. The cancellation of the forces is done at every step, and along all 3 directions of space, x, y, and z, due to the use of *0 0 0*. The two *velocity* commands set the initial velocities along x, y, and z to 0 for the atoms of *carbon_top* and *carbon_bot*, respectively.

As a consequence of these last four commands, the atoms of the edges will remain immobile during the simulation (or at least they would if no other command was applied to them).

On imposing a constant velocity to a system

The *velocity set* commands impose the velocity of a group of atoms at the start of a run, but does not enforce the velocity during the entire simulation. When *velocity set* is used in combination with *setforce 0 0 0*, as is the case here, the atoms wont feel any force during the entire simulation. According to the Newton equation, no force means no acceleration, meaning that the initial velocity will persist during the entire simulation, thus producing a constant velocity motion.

2.1.5 Data extraction

Next, in order to measure the strain and stress suffered by the CNT, let us extract the distance L between the two edges as well as the force applied on the edges.

```
variable L equal xcm(carbon_top,z)-xcm(carbon_bot,z)
fix at1 all ave/time 10 10 100 v_L file output_cnt_length.dat
fix at2 all ave/time 10 10 100 f_mysf1[1] f_mysf2[1] &
    file output_edge_force.dat
```

Let us also add a command to print the atom coordinates in a *lammpstrj* file every 1000 steps.

```
dump mydmp all atom 1000 dump.lammpstrj
```

Finally, let us check the temperature of the non-frozen group over time by printing it using a *fix ave/time* command:

```
fix at3 all ave/time 10 10 100 c_Tmid &
    file output_temperature_middle_group.dat
```

About extracting quantity from variable compute or fix

Notice that the values of the force on each edge are extracted from the two *fix setforce mysf1* and *mysf2*, simply by calling them using *f_*, the same way variables are called using *v_* and computes are called using *c_*.

Let us run a small equilibration step to bring the system to the required temperature before applying any deformation:

```
thermo 100
thermo_modify temp Tmid

timestep 1.0
run 5000
```

With the *thermo_modify* command, we specify to LAMMPS that we want the temperature T_{mid} to be printed in the terminal, not the temperature of the entire system (because of the frozen edges, the temperature of the entire system is not relevant).

Let us impose a constant velocity deformation to the CNT by combining the *velocity set* command with previously defined *fix setforce*. Add the following lines in the *input.lammps* file, right after the last *run 5000* command:

```
# 2*0.0005 A/fs = 0.001 A/fs = 100 m/s
velocity carbon_top set 0 0 0.0005
velocity carbon_bot set 0 0 -0.0005
run 10000
```

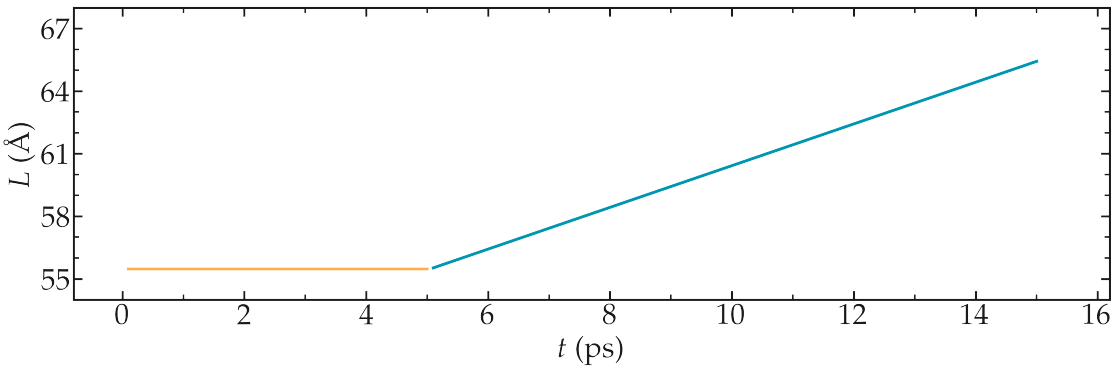


Figure: Evolution of the lenght of the CNT with time. The CNT starts deforming at $t = 5\,\text{ps}$.

The chosen velocity for the deformation is $100\,\text{m/s}$.

The energy, which can be accessed from the log file, shows a non-linear increase with time once the deformation starts, which is extected from the typical dependency of bond energy with bond distance $U_b = k_b\,(r - r_0)^2$.

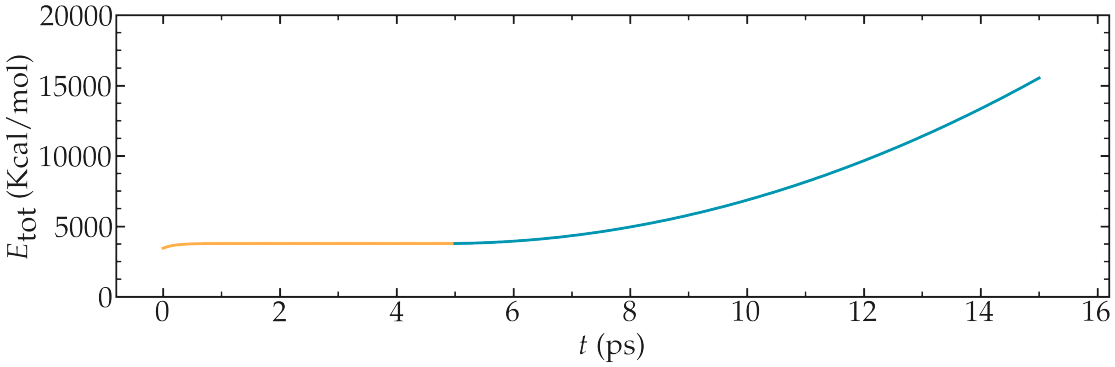


Figure: Evolution of the total energy of the system with time. The CNT starts deforming at $t = 5\,\text{ps}$.

As always, is it important to control that the simulation behaved as expected by opening the *dump.lammpstrj* file with VMD.

2.2 Breakable bonds

When using classical force field, as we just did, the bonds between atoms are non-breakable. Let us perform a similar simulation, but this time using a reactive force field instead, allowing for the bonds to break if the applied deformation is large enough.

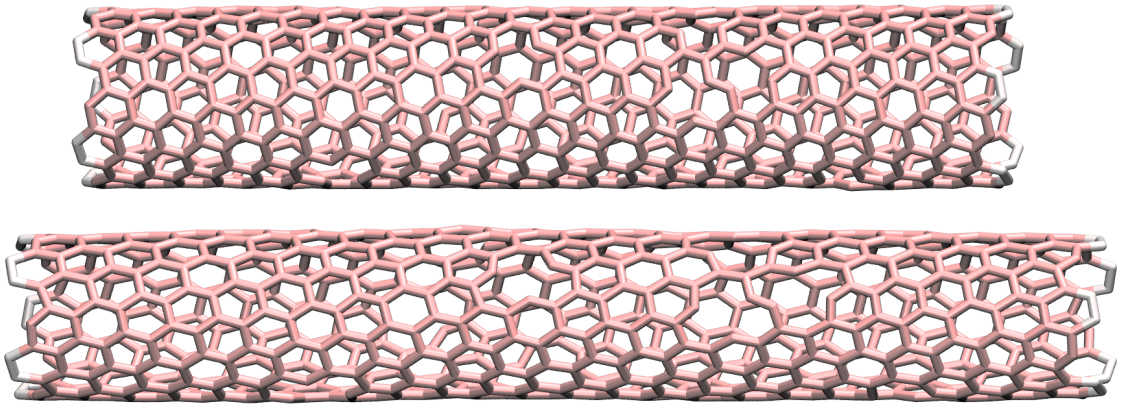


Figure: CNT before (top) and after (bottom) deformation. See the corresponding video.

2.2.1 Input file initialization

Create a second folder named *breakable-bonds/* next to *unbreakable-bonds/*, and create a new input file in it called *input.lammps*. Type into input.lammps:

```
# Initialisation
variable T equal 300

units metal
atom_style atomic
boundary p p p
pair_style airebo 2.5 1 1
```

A first difference with the previous part is the unit system, here *metal* instead of *real*, a choice that is imposed by the AIREBO force field. A second difference is the use of the *atom_style atomic* instead of *molecular*, single no explicit bond information is required with AIREBO.

About metal units

With the *metal* units system of LAMMPS, the time is in pico second, distances are in Angstrom, and the energy is in eV.

2.2.2 Adapt the topology file

Since *bond*, *angle*, and *dihedral* do not need to be explicitely set when using AIREBO, some small changes needs to be made to the previously generated *.data* file.

Duplicate the previous file *cnt_molecular.data*, name the copy *cnt_atom.data*, place it within *breakable-bonds/*. Then, remove all bond, angle, and dihedral information from *cnt_atom.data*. Also remove the second column of the *Atoms* table, so that the *cnt_atom.data* looks like the following:

```
700 atoms
1 atom types
-40.000000 40.000000  xlo xhi
-40.000000 40.000000  ylo yhi
-12.130411 67.869589  zlo zhi

Masses

1 12.010700 # CA

Atoms # atomic

1 1 5.162323 0.464617 8.843235 # CA CNT
2 1 4.852682 1.821242 9.111212 # CA CNT
(...)
```

In addition, remove the *Bonds* table that is places right after the *Atoms* table (near line 743), as well as the *Angles*, *Dihedrals*, and *Impropers* tables. The last lines of the file should look like that:

```
(...)
697 1 4.669892 -2.248901 45.824036 # CA CNT
698 1 5.099893 -0.925494 46.092010 # CA CNT
699 1 5.162323 -0.464617 47.431896 # CA CNT
700 1 5.099893 0.925494 47.699871 # CA CNT
```

Alternatively, you can also download the file I did generate by clicking here.

2.2.3 Use of AIREBO potential

Then, let us import the LAMMPS data file, and set the pair coefficients by adding the following lines to *input.lammps*

```
# System definition
read_data cnt_atom.data
pair_coeff * * CH.airebo C
```

Here, there is one single atom type. We impose this type to be carbon by using the the letter *C*.

Setting AIREBO pair coefficients

In case of multiple atom types, one has to adapt the *pair_coeff* command. If there are 2 atom types, and both are carbon, it would read: *pair_coeff CH.airebo C C*. If atoms of type 1 are carbon, and atoms type 2 are hydrogen, then *pair_coeff CH.airebo C H*.

The *CH.airebo* file can be downloaded by clicking here, and must be placed within the *breakable-bonds/* folder. The rest of the *input.lammps* is very similar to the previous one:

```
change_box all x final -40 40 y final -40 40 z final -60 60

group carbon_atoms type 1
variable carbon_xcm equal -1*xcm(carbon_atoms,x)
variable carbon_ycm equal -1*xcm(carbon_atoms,y)
variable carbon_zcm equal -1*xcm(carbon_atoms,z)
displace_atoms carbon_atoms move ${carbon_xcm} ${carbon_ycm} ${carbon_zcm}

variable zmax equal bound(carbon_atoms,zmax)-0.5
variable zmin equal bound(carbon_atoms,zmin)+0.5
region rtop block INF INF INF INF ${zmax} INF
region rbot block INF INF INF INF INF ${zmin}
region rmid block INF INF INF INF ${zmin} ${zmax}

group carbon_top region rtop
group carbon_bot region rbot
group carbon_mid region rmid

variable zmax_del equal ${zmax}-2
variable zmin_del equal ${zmin}+2
region rdel block INF INF INF INF ${zmin_del} ${zmax_del}
group rdel region rdel
delete_atoms random fraction 0.02 no rdel NULL 482793

reset_atoms id sort yes
velocity carbon_mid create ${T} 48455 mom yes rot yes
fix mynve all nve
compute Tmid carbon_mid temp
fix myber carbon_mid temp/berendsen ${T} ${T} 0.1
fix_modify myber temp Tmid
```

Note that a large distance of 120 Angstroms was used for the box size along the z axis, to allow for larger deformation. In additoin, the *change_box* command was placed before the *displace_atoms* to avoid issue with the CNT crossing the edge of the box.

2.2.4 Start the simulation

Here, let us impose a constant velocity deformation using the atoms of one edge, while maintaining the other edge fix. Do to so, one needs to cancel the forces (thus the acceleration) on the atoms of the edges using the *setforce* command, and set the value of the velocity along the z direction.

First, as an equilibration step, let us set the velocity to 0 for the atoms of both edges. Let us fully constrain the edges. Add the following lines to LAMMPS:

```
fix mysf1 carbon_bot setforce 0 0 0
fix mysf2 carbon_top setforce 0 0 0
velocity carbon_bot set 0 0 0
velocity carbon_top set 0 0 0

variable L equal xcm(carbon_top,z)-xcm(carbon_bot,z)
fix at1 all ave/time 10 10 100 v_L file output_cnt_length.dat
fix at2 all ave/time 10 10 100 f_mysf1[1] f_mysf2[1] &
    file output_edge_force.dat

dump mydmp all atom 1000 dump.lammpstrj

thermo 100
thermo_modify temp Tmid

timestep 0.0005
run 5000
```

Note the relatively small timestep of 0.0005 ps used. Reactive force field usually requires smaller timestep than classical one. When running *input.lammps* with LAMMPS, you can see that the temperature deviates from the target temperature of 300 K at the start of the equilibration, but that after a few steps it reaches the target value:

```
Step  Temp          E_pair         E_mol          TotEng         Press
0     300          -5084.7276      0             -5058.3973     -1515.7017
100   237.49462    -5075.4114      0             -5054.5671     -155.05545
200   238.86589    -5071.9168      0             -5050.9521     -498.15029
300   220.04074    -5067.1113      0             -5047.7989     -1514.8516
400   269.23434    -5069.6565      0             -5046.0264     -174.31158
500   274.92241    -5068.5989      0             -5044.4696     -381.28758
600   261.91841    -5065.985       0             -5042.9971     -1507.5577
700   288.47709    -5067.7301      0             -5042.4111     -312.16669
800   289.85177    -5066.5482      0             -5041.1086     -259.84893
900   279.34891    -5065.0216      0             -5040.5038     -1390.8508
1000  312.27343    -5067.6245      0             -5040.217      -465.74352
(...)
```

2.2.5 Launch the deformation

After equilibration, let us set the velocity to 15 m/s and run for a longer duration that previously. Add the following lines into *input.lammps*:

```
# 0.15 A/ps = 15 m/s
velocity carbon_top set 0 0 0.15
run 280000
```

The CNT should break around the step 250000. If not, run for a longer time.

When looking at the *lammpstrj* file using VMD, you will see the bonds breaking. Use the *DynamicBonds* representation to properly vizualise the bond breaking.

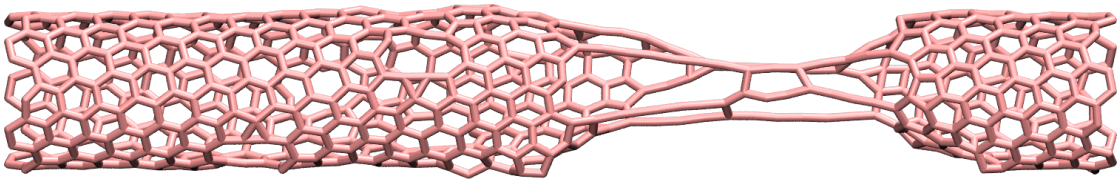


Figure: CNT with broken bonds. See the corresponding video.

About bonds in VMD

Note that VMD guesses bonds based on the distances between atoms, and not based on the presence of actual bonds between atoms in the LAMMPS simulation. Therefore what is seen in VMD can sometimes be misleading.

Looking at the evolution of the energy again, one can see the energy increasing with the deformation, before completely relaxing when the CNT finally breaks.

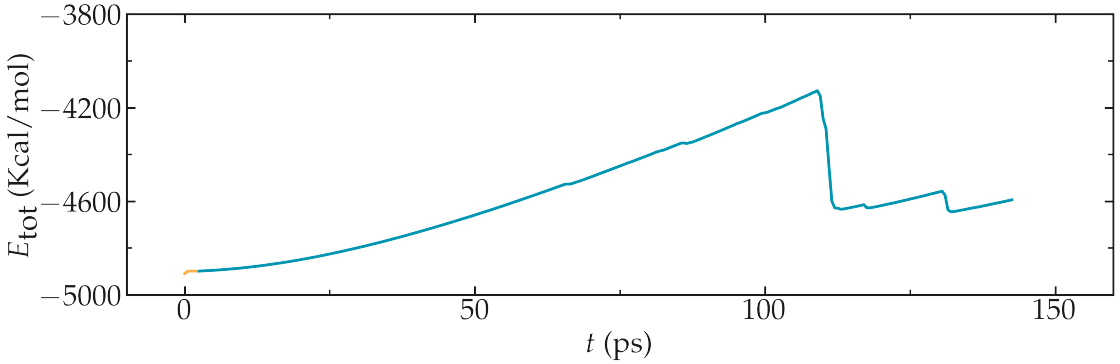


Figure: Evolution of the total energy of the system with time.

You can access all the input scripts and data files that are used in these tutorials from the inputs folder on Github. This repository also contains the full solutions to the exercises.

There is a follow-up to this CNT tutorial as a MDAnalysis tutorial, where a post-mortem analysis is performed using Python.

2.3 Going further with exercises

Each exercise comes with a proposed solution, see Solutions to the exercises.

2.3.1 Plot the strain-stress curves

Adapt the current scripts and extract the strain-stress curves for the two breakable and unbreakable CNTs:

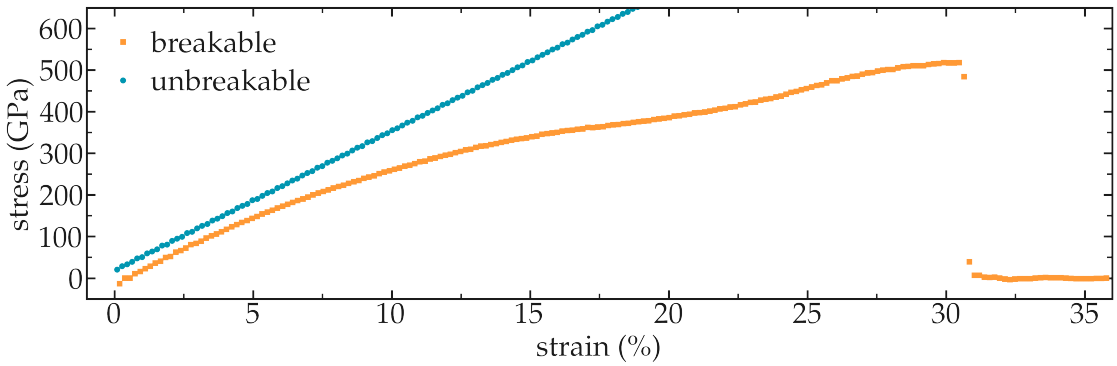


Figure: Strain-stain curves for the two CNTs, breakable and unbreakable.

2.3.2 Solve the flying ice cube artifact

The flying ice cube effect is one of the most famous artifact of molecular simulations. Download this seemingly simple input, that was inspired by the first part of the tutorial, and simplified by removing some of it. Run the input with this data file and this parameter file.

When you run this simulation using LAMMPS, you should see that the temperature is very close to 300 K, as expected.

```
Step    Temp         E_pair       E_mol        TotEng       Press
0       327.4142     589.20707    1980.6012    3242.2444    60.344754
1000    300.00184    588.90015    1980.9013    3185.9386    51.695282
(...)
```

However, if you look at the system using VMD, the atoms are not moving.

Can you identify the origin of the issue, and fix the input?

2.3.3 Insert gas in the carbon nanotube

Modify the input from the unbreakable CNT, and add atoms of argon within the CNT.

Use the following *pair_coeff* for the argon, and a mass of *39.948*:

```
pair_coeff 2 2 0.232 3.3952
```

2.3.4 Make a membrane of CNTs

Replicate the CNT along the x and y direction, and equilibrate the system to create an infinite membrane made of multiple CNTs.

Apply a shear deformation along xy.

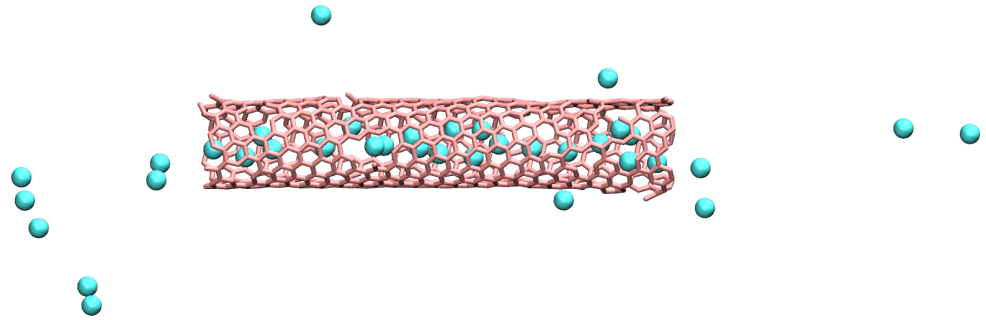


Figure: Argon atoms in a CNT. See the corresponding video.

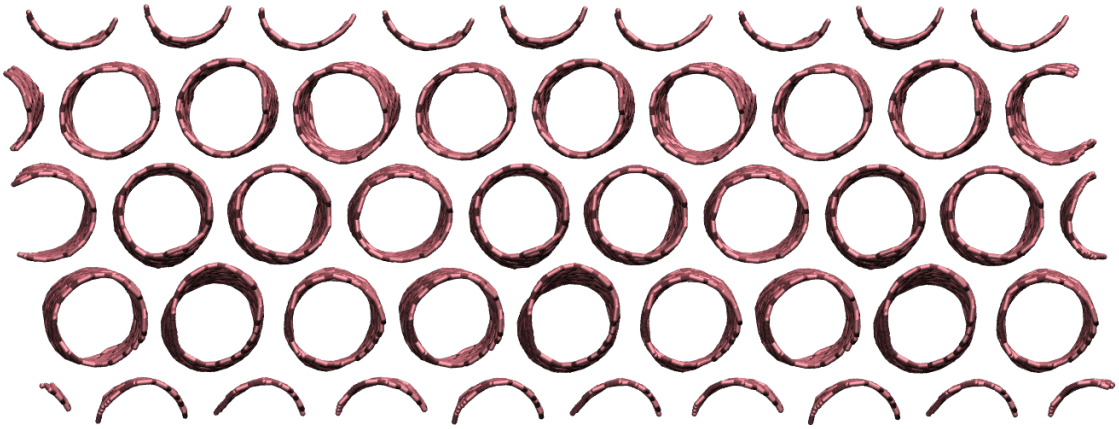


Figure: Multiple carbon nanotubes forming a membrane.

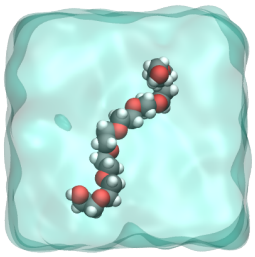
Hint

The box must be converted to triclinic to support deformation along *xy*.

3

Polymer in water

Solvating and stretching a small polymer molecule



The goal of this tutorial is to use LAMMPS and solvate a small hydrophilic polymer (PEG - PolyEthylene Glycol) in a reservoir of water.

An all-atom description is used for both PEG and water, and the long range Coulomb interactions are solved using the PPPM solver. Once the PEG and water system is properly equilibrated at the desired temperature and pressure, a constant stretching force is applied to both ends of the polymer, and the evolution of its length is measured as a function of the time.

This tutorial was inspired by a publication by Liese and coworkers, in which molecular dynamics simulations are compared with force spectroscopy experiments.

If you are completely new to LAMMPS, I recommend you to follow this tutorial on a simple Lennard Jones fluid first.

3.1 Preparing water and PEG separately

In this tutorial, the water is being prepared separately from the PEG molecule. The PEG and water will be merged later.

3.1.1 The water

As a first step, a rectangular box of water is created and equilibrated at ambient temperature and ambient pressure. Create a folder named *pureH2O/*. Inside this folder, create an empty text file named *input.lammps*. Copy the following lines in it:

```
units real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style harmonic
pair_style lj/cut/coul/long 12
kspace_style pppm 1e-5
special_bonds lj 0.0 0.0 0.5 coul 0.0 0.0 1.0 angle yes
```

With the unit style *real*, masses are in grams per mole, distances in Ångstroms, time in femtoseconds, energies in Kcal/mole. With the *atom_style full*, each atom is a dot with a mass and a charge that can be linked by bonds, angles, dihedrals and impropers. The *bond_style*, *angle_style*, and *dihedral_style* commands define the potentials for the bonds, angles, and dihedrals used in the simulation, here *harmonic*.

Always refer to the LAMMPS documentation if you have doubts about the potential used by LAMMPS. For instance, this page gives the expression for the harmonic angular potential.

Finally, the *special_bonds* command cancels the Lennard-Jones interactions between the closest atoms of a same molecule.

About *special bonds*

Usually, molecular dynamics force fields are parametrized assuming that the first neighbors within a molecule do not interact directly though LJ or Coulomb potential. Here, since we use *lj 0.0 0.0 0.5* and *coul 0.0 0.0 1.0*, the first and second neighbors in a molecule only interact through direct bond interactions. For the third neighbor (here third neighbor only concerns the PEG molecule, not the water), only half of the LJ interaction will be taken into account, and the full Coulomb interaction will be used.

With the *pair_style* named *lj/cut/coul/long*, atoms interact through both a Lennard-Jones (LJ) potential and Coulombic interactions. The value of 12 Å is the cutoff.

About cutoff in molecular dynamics

The cutoff of 12 Å applies to both LJ and Coulombic interactions, but in a different way. For LJ *cut* interactions, atoms interact with each others only if they are separated by a distance smaller than the cutoff. For Coulombic *long*, interactions between atoms closer than the cutoff are computed directly, and interactions between atoms outside that cutoff are computed in the reciprocal space.

Finally the kspace command defines the long-range solver for the (long) Coulombic interactions. The *pppm* style refers to particle-particle particle-mesh.

About PPPM

From Luty and van Gunsteren: 'The PPPM method is based on separating the total interaction between particles into the sum of short-range interactions, which are computed by direct particle-particle summation, and long-range interactions, which are calculated by solving Poisson's equation using periodic boundary conditions (PBCs).'

Then, let us create a 3D simulation box of dimensions $3 \times 3 \times 3$ nm^3, and make space for 9 atom types (2 for the water molecule, and 7 for the PEG molecule), 6 bond types, 15 angle types, and 3 dihedrals types. Copy the following lines into *input.lammps*:

```
region box block -15 15 -15 15 -15 15
create_box 9 box &
bond/types 6 &
angle/types 15 &
dihedral/types 3 &
extra/bond/per/atom 2 &
extra/angle/per/atom 1 &
extra/special/per/atom 2
```

About extra per atom commands

The *extra/x/per/atom* commands are here for memory allocation. These commands ensure that enough memory space is left for a certain number of attribute for each atom. We wont worry about those commands in this tutorial, just keep that in mind if one day you see the following error message *ERROR: Molecule topology/atom exceeds system topology/atom*.

Let us create a *PARM.lammps* file containing all the parameters (masses, interaction energies, bond equilibrium distances, etc). In *input.lammps*, add the following line:

```
include ../PARM.lammps
```

Then, download and save the parameter file next to the *pureH2O/* folder.

Within *PARM.lammps*, the *mass* and *pair_coeff* of atoms of type 8 and 9 are for water, while those of atoms type 1 to 7 are for the PEG molecule. Similarly, the *bond_coeff 6* and *angle_coeff 15* are for water, while all the other parameters are for the PEG.

Let us create water molecules. To do so, let us define what a water molecule is using a molecule *template* called *FlexibleH2O.txt*, and then randomly create 350 molecules. Add the following lines into *input.lammps*:

```
molecule h2omol FlexibleH2O.txt
create_atoms 0 random 350 45615 NULL mol &
    h2omol 14756 overlap 1 maxtry 50
```

The *overlap 1* option of the *create_atoms* command ensures that no atoms are placed exactly at the same position, as this would cause the simulation to crash. The *maxtry 50* asks LAMMPS to try at most 50 times to insert the molecules, which is useful in case some insertion attempts are rejected due to overlap. In some case, depending on the system and on the values of *overlap* and *maxtry*, LAMMPS may not create the desired number of molecules. Always check the number of created atoms in the *log* file after starting the simulation:

```
Created 1050 atoms
```

When LAMMPS fails to create the desired number of molecules, a WARNING appears in the *log* file.

The molecule template named *FlexibleH2O.txt* can be downloaded and saved in the *pureH2O/* folder. This template contains the necessary structural information of a water molecule, such as the number of atoms, the id of the atoms that are connected by bonds, by angles, etc.

Then, let us organize the atoms of type 8 and 9 of the water molecules in a group named *H2O*, and then perform a small energy minimization. The energy minimization is mandatory here given the small *overlap* value of 1 Angstrom chosen in the *create_atoms* command. Add the following lines to *input.lammps*:

```
group H2O type 8 9
minimize 1.0e-4 1.0e-6 100 1000
reset_timestep 0
```

The *reset_timestep* command is optional. It is used here because the *minimize* command is usually performed over an arbitrary number of steps.

Let us use the *fix npt* to control both the temperature and the pressure of the system, by adding the following line to *input.lammps*:

```
fix mynpt all npt temp 300 300 100 iso 1 1 1000
```

The *fix npt* allows us to impose both a temperature of 300 K (with a damping constant of 100 fs), and a pressure of 1 atmosphere (with a damping constant of 1000 fs). With the *iso* keyword, the three dimensions of the box will be re-scaled simultaneously.

Let us print the atom positions in a *.lammpstrj* file every 1000 steps (i.e. 1 ps), print the temperature volume, and density every 100 steps in 3 separate data files, and print the information in the terminal every 1000 steps:

```
dump mydmp all atom 1000 dump.lammpstrj
variable mytemp equal temp
variable myvol equal vol
fix myat1 all ave/time 10 10 100 v_mytemp file temperature.dat
fix myat2 all ave/time 10 10 100 v_myvol file volume.dat
variable myoxy equal count(H2O)/3
variable mydensity equal ${myoxy}/v_myvol
fix myat3 all ave/time 10 10 100 v_mydensity file density.dat
thermo 1000
```

The variable *myoxy* corresponds to the number of atoms divided by 3, i.e. the number of molecules.

On calling variables in LAMMPS

Both dollar sign and underscore can be used to call a previously defined variable. With the dollar sign, the initial value of the variable is returned, while with the underscore, the instantaneous value of the variable is returned. To probe the temporal evolution of a variable with time, the underscore must be used.

Finally, let us set the timestep to 1.0 fs, and run the simulation for 50 ps by adding the following lines to *input.lammps*:

```
timestep 1.0
run 50000

replicate 3 1 1

write_data H2O.data
```

The *replicate* command is used to replicate the final cubic box three times along the *x* direction, thus creating a rectangular box of water just before the final state is written into *H2O.data*.

If you open the *dump.lammpstrj* file using VMD, you should see the system quickly reaching its equilibrium volume and density.

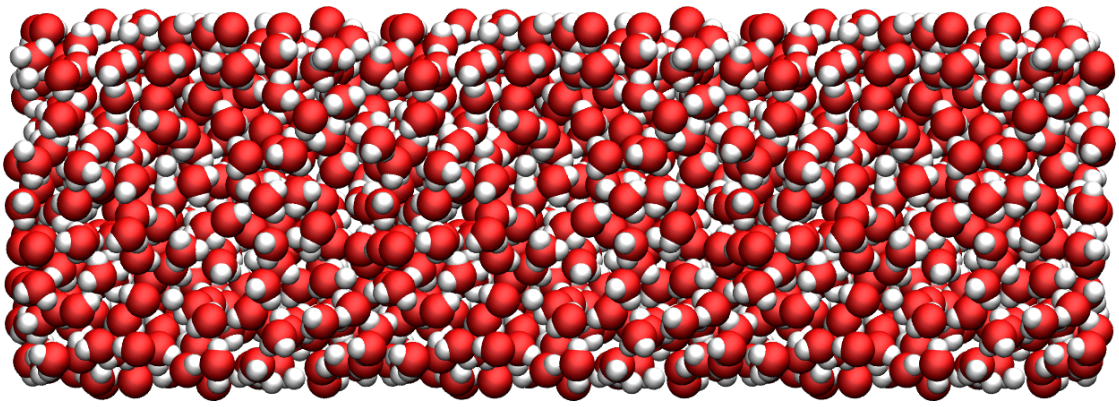


Figure: Water reservoir after equilibration and after the *replicate 3 1 1* command. Oxygen atoms are in red, and hydrogen atoms in white.

You can also open the *density.dat* file to ensure that the system converged toward an equilibrated liquid water system during the 50 ps of simulation.

If needed, you can download the water reservoir I have equilibrated and continue with the tutorial.

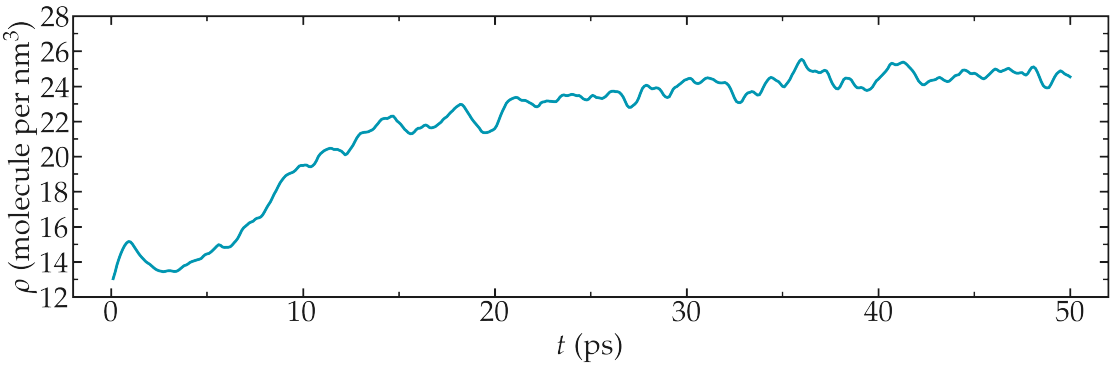


Figure: Evolution of the density of water with time. The density ρ reaches a plateau after ≈ 30 ps.

3.1.2 The PEG molecule

Now that the water box is ready, let us prepare the PEG molecule in an empty box. Create a second folder next to *pureH2O/*, call it *singlePEG/*, and create a new blank file called *input.lammps* in it. Copy the same first lines as previously:

```
units real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style harmonic
pair_style lj/cut/coul/long 12
kspace_style pppm 1e-5
special_bonds lj 0.0 0.0 0.5 coul 0.0 0.0 1.0 angle yes dihedral yes
```

Let us read the original positions for the atoms of the PEG molecule, as well as the same parameter file as previously, by adding the following lines to *input.lammps*:

```
read_data init.data
include ../PARM.lammps
```

Download the init.data file and save it in the *singlePEG/* folder. It contains the initial parameters of the PEG molecules (atoms, bonds, charges, etc.), and was downloaded from the ATB repository.

Let us print the atom positions and thermodynamic information very frequently (because we anticipate that the energy minimization will be short). Add the following lines to *input.lammps*:

```
dump mydmp all atom 10 dump.lammpstrj
thermo 1
```

Next, let us perform a minimization of energy. Add the following line to *input.lammps*:

```
minimize 1.0e-4 1.0e-6 100 1000
```

After the minimization, the high frequency dump command is cancelled, and a new dump command with lower frequency is used (see below). We also reset the time to 0 with *reset_timestep* command:

```
undump mydmp
reset_timestep 0
```

The PEG is then equilibrated in the NVT ensemble (fix NVE + temperature control = NVT). No box relaxation is required as the PEG is in vacuum:

```
fix mynve all nve
fix myber all temp/berendsen 300 300 100
```

Let us print the temperature in a file by also adding the following lines to *input.lammps*:

```
dump mydmp all atom 1000 dump.lammpstrj
dump_modify mydmp append yes
thermo 1000
variable mytemp equal temp
fix myat1 all ave/time 10 10 100 v_mytemp file temperature.dat
```

The *dump_modify* ensures that the coordinates are written in the already existing *dump.lammpstrj* file. Finally let us run the simulation for a very short time (10 ps) by adding the following lines to *input.lammps*:

```
timestep 1.0
run 10000
write_data PEG.data
```

If you open the *dump.lammpstrj* file using VMD, you can see the PEG molecule gently equilibrating until reaching a reasonable state.

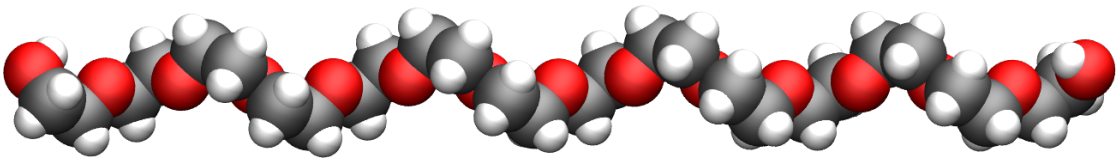


Figure: The PEG molecule in vacuum. The carbon atoms are in gray, the oxygen atoms in red, and the hydrogen atoms in white. See the corresponding video.

Alternatively, you can download the PEG molecule I have equilibrated and continue with the tutorial.

3.2 Solvated PEG

Once both the water and the PEG are equilibrated separately, we can safely merge the two systems before performing the pull experiment on the polymer.

3.2.1 Mixing the PEG with water

To merge the two systems, let us:

- (1) import both previously generated data files (PEG.data and H2O.data) into the same simulation,
- (2) delete the water molecules that are overlapping with the PEG molecule, and
- (3) re-equilibrate the new system.

Create a third folder alongside *pureH2O/* and *singlePEG/*, and call it *mergePEGH2O/*. Create a new blank file in it, called *input.lammps*. Within *input.lammps*, copy the same first lines as previously:

```
units real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style harmonic
pair_style lj/cut/coul/long 12
kspace_style pppm 1e-5
special_bonds lj 0.0 0.0 0.5 coul 0.0 0.0 1.0 angle yes dihedral yes
```

Then, import the two previously generated data files *H2O.data* and *PEG.data*, as well as the *PARM.lammps* file by adding that to *input.lammps*:

```
read_data ../pureH2O/H2O.data &
extra/bond/per/atom 2 &
extra/angle/per/atom 5 &
extra/dihedral/per/atom 9
read_data ../singlePEG/PEG.data add append shift 25 0 0
include ../PARM.lammps
```

When using the *read_data* command more than once, one needs to use the *add append* keyword. When doing so, the simulation box is initialized by the first *read_data* only, and the second *read_data* only imports additional atoms.

The *extra/x/per/atom* commands are again here for memory allocation. The *shift 25 0 0* that is applied to the polymer is there to recenter the polymer in the rectangular box by shifting its position by 25 Angstroms along the *x* axis.

Let us create 2 groups to differentiate the PEG from the H2O, by adding the following lines to *input.lammps*:

```
group H2O type 8 9
group PEG type 1 2 3 4 5 6 7
```

Water molecules that are overlapping with the PEG must be deleted to avoid future crashing. Add the following line to *input.lammps*:

```
delete_atoms overlap 2.0 H2O PEG mol yes
```

Here, the value of 2 Angstroms for the overlap cutoff was fixed arbitrarily, and can be chosen through trial and error. If the cutoff is too small, the simulation will crash. If the cutoff it too large, too many water molecules will unnecessarily be deleted.

Finally, let us use the *fix NPT* to control the temperature, as well as the pressure by allowing the box size to be rescalled along the *x* axis:

```
fix mynpt all npt temp 300 300 100 x 1 1 1000
timestep 1.0
```

Once more, let us dump the atom positions and a few information about the evolution of the simulation:

```
dump mydmp all atom 100 dump.lammpstrj
thermo 100
variable mytemp equal temp
variable myvol equal vol
fix myat1 all ave/time 10 10 100 v_mytemp file temperature.dat
fix myat2 all ave/time 10 10 100 v_myvol file volume.dat
```

Let us also print the total enthalpy:

```
variable myenthalpy equal enthalpy
fix myat3 all ave/time 10 10 100 v_myenthalpy file enthalpy.dat
```

Finally, let us perform a short equilibration and print the final state in a data file. Add the following lines to the data file:

```
run 10000
write_data mix.data
```

If you open the *dump.lammpstrj* file using VMD, or have a look at the evolution of the volume in *volume.dat*, you should see that the box dimension slightly evolves along *x* to accomodate the new configuration.

3.2.2 Stretching the PEG molecule

Here, a constant forcing is applied to the two ends of the PEG molecule until it stretches. Create a new folder next to the 3 previously created folders, call it *pullonPEG/* and create a new input file in it called *input.lammps*.

First, let us create a variable *f0* corresponding to the magnitude of the force we are going to apply. The force magnitude is chosen to be large enough to overcome the thermal agitation and the entropic contribution from both water and PEG molecules (it was chosen by trial and error). Copy in the *input.lammps* file:

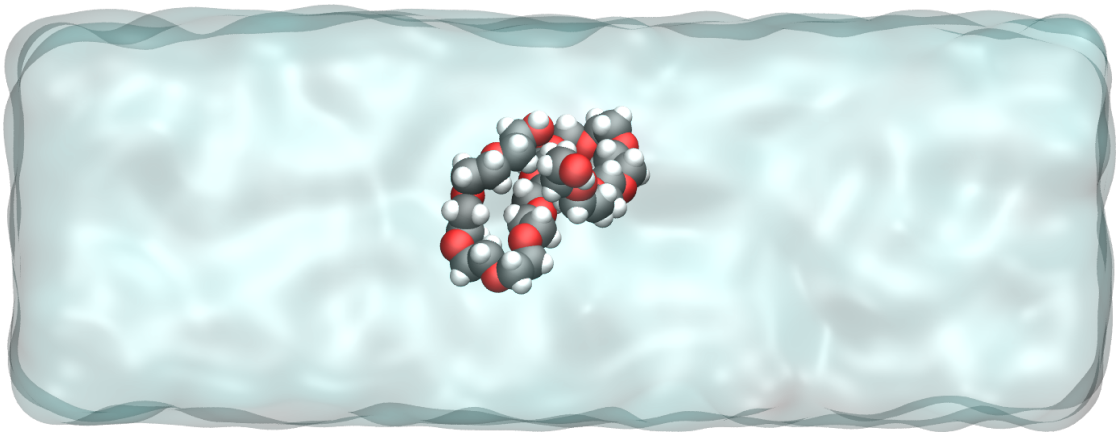


Figure: A single PEG molecule in water. Water molecules are represented as a transparent continuum field for clarity.

```
variable f0 equal 5
```

Note that 1 kcal/mol/Å corresponds to 67.2 pN. Then, copy the same lines as previouly,:

```
units real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style harmonic
pair_style lj/cut/coul/long 12
kspace_style pppm 1e-5
special_bonds lj 0.0 0.0 0.5 coul 0.0 0.0 1.0 angle yes dihedral yes
```

Start the simulation from the equilibrated PEG and water system, and include again the parameter file by adding the following lines to the *input.lammps*:

```
read_data ../mergePEGH2O/mix.data
include ../PARM.lammps
```

Then, let us create 4 atom groups: H2O and PEG (as previously), as well as 2 groups containing only the 2 oxygen atoms of types 6 and 7, respectively. Atoms of types 6 and 7 correspond to the oxygen atoms located at the ends of the PEG molecule, which we are going to use to pull on the PEG molecule. Add the following lines to the *input.lammps*:

```
group H2O type 8 9
group PEG type 1 2 3 4 5 6 7
group topull1 type 6
group topull2 type 7
```

Let us add the *dump* command again to print the atom positions:

```
dump mydmp all atom 1000 dump.lammpstrj
```

Let us use a simple thermostating for all atoms by adding the following lines to *input.lammps*:

```
timestep 1.0
fix mynvt all nvt temp 300 300 100
```

Let us also print the end-to-end distance of the PEG, here defined as the distance between the groups *topull1* and *topull2*, as well as the temperature of the system by adding the following lines to *input.lammps*:

```
variable mytemp equal temp
fix myat1 all ave/time 10 10 100 v_mytemp file output-temperature.dat
variable x1 equal xcm(topull1,x)
variable x2 equal xcm(topull2,x)
variable y1 equal xcm(topull1,y)
variable y2 equal xcm(topull2,y)
variable z1 equal xcm(topull1,z)
variable z2 equal xcm(topull2,z)
variable delta_r equal sqrt((v_x1-v_x2)^2+(v_y1-v_y2)^2+(v_z1-v_z2)^2)
fix myat2 all ave/time 10 10 100 v_delta_r &
    file output-end-to-end-distance.dat
thermo 1000
```

Finally, let us run the simulation for 30 ps without any external forcing:

```
run 30000
```

This first run serves a benchmark to quantify the changes induced by the forcing. Then, let us apply a forcing on the 2 oxygen atoms using two *add_force* commands, and run for an extra 30 ps:

```
fix myaf1 oxygen_end1 addforce ${f0} 0 0
fix myaf2 oxygen_end2 addforce -${f0} 0 0
run 30000
```

If you open the *dump.lammpstrj* file using *VMD*, you should see that the PEG molecule eventually aligns in the direction of the force.

The evolution of the end-to-end distance over time shows the PEG adjusting to the external forcing:

You can access all the input scripts and data files that are used in these tutorials from the inputs folder on Github. This repository also contains the full solutions to the exercises.

3.3 Going further with exercises

Each exercise comes with a proposed solution, see Solutions to the exercises.

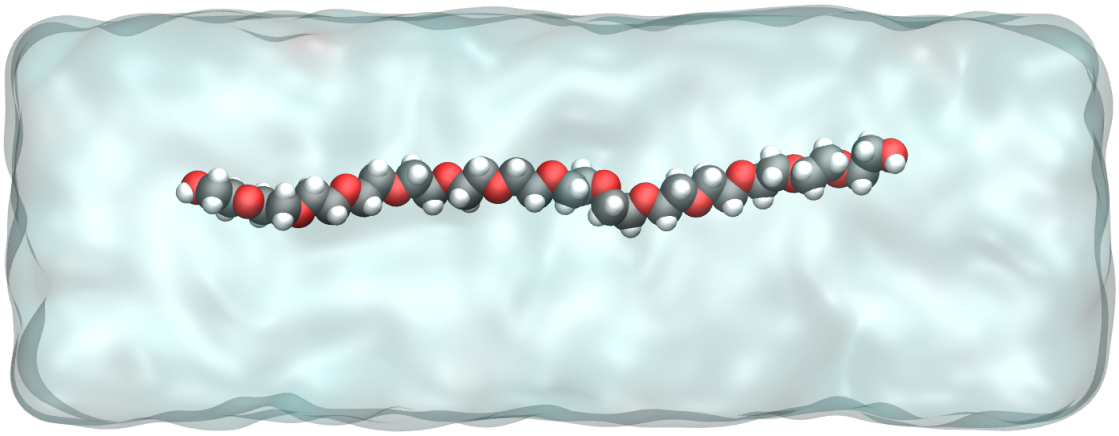


Figure: PEG molecule streched along the x direction in water. Water molecules are represented as a transparent continuum field for clarity. See the corresponding video.

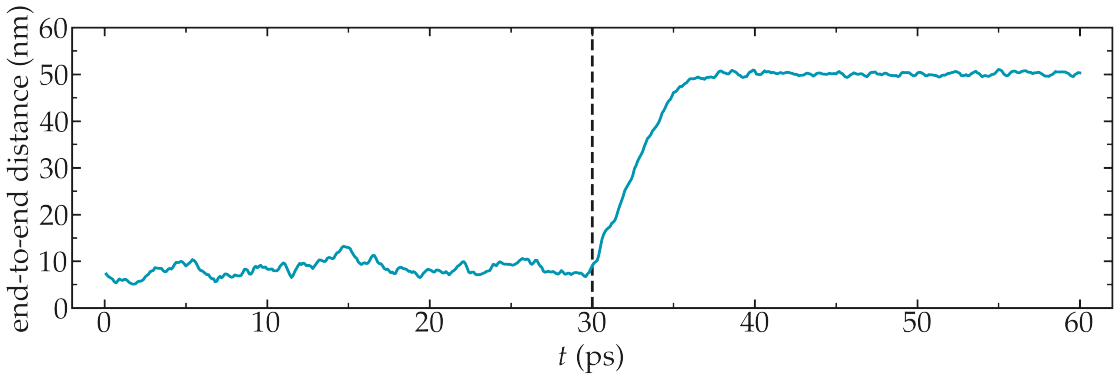


Figure: Evolution of the end-to-end distance of the PEG molecule with time. The forcing starts at $t = 30$ ps.

3.3.1 Extract the radial distribution function

Extract the radial distribution functions (RDF or $g(r)$) between the oxygen atom of the water molecules and the oxygen atom from the PEG molecule. Compare the rdf before and after the force is applied to the PEG.

Note the difference in the structure of the water before and after the PEG molecule is being stretched. This effect is described in the 2017 publication by Liese et al.

3.3.2 Add salt to the system

Realistic systems usually contain ions. Let us add some Na^+ and Cl^- ions to our current PEG-water system.

Add some Na^+ and Cl^- ions to the mixture using the method of your choice. Na^+ ions are

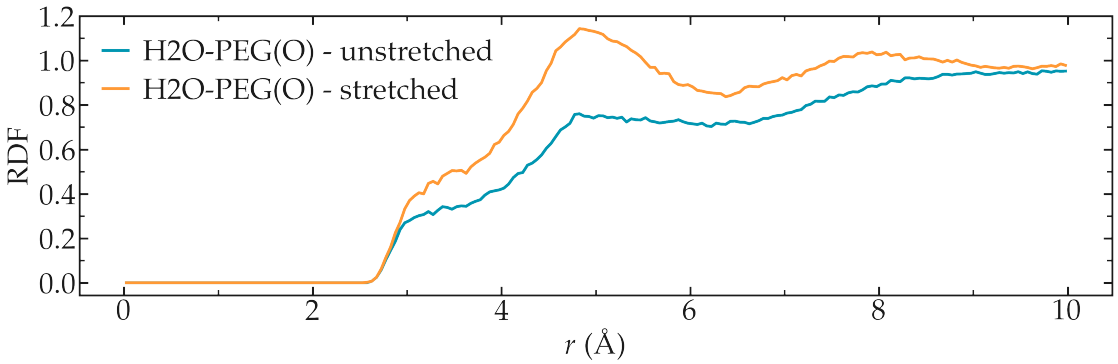


Figure: Radial distribution function between the oxygen atoms of water, as well as between the oxygen atoms of water and the oxygen atoms of the PEG molecule.

characterised by their mass $m = 22.98\,\text{g/mol}$, their charge $q = +1\,e$, and Lennard-Jones parameters, $\epsilon = 0.0469\,\text{kcal/mol}$, $\sigma = 0.243\,\text{nm}$, and Cl^- ions by their mass $m = 35.453\,\text{g/mol}$, charge $q = -1\,e$ and Lennard-Jones parameters, $\epsilon = 0.15\,\text{kcal/mol}$, $\sigma = 0.4045\,\text{nm}$.

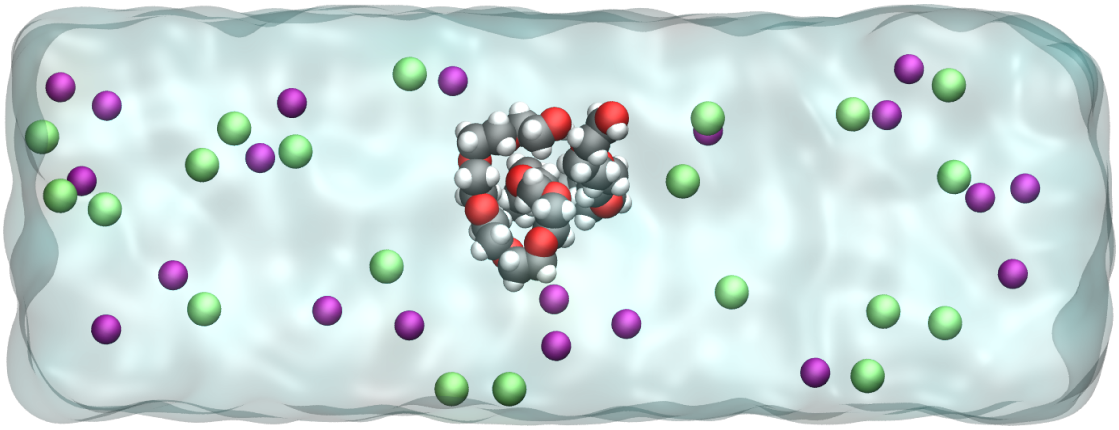


Figure: A PEG molecule in electrolyte, with Na^+ ions in purple and Cl^- ions in cyan.

3.3.3 Evaluate the deformation of the PEG

Once the PEG is fully stretched, its structure differs from the unstretched case. The deformation can be probed by extracting the typical intra-molecular parameters, such as the typical angles of the dihedrals.

Extract the histograms of the anglar distribution of the PEG dihedrals in the absence and in the presence of stretching.

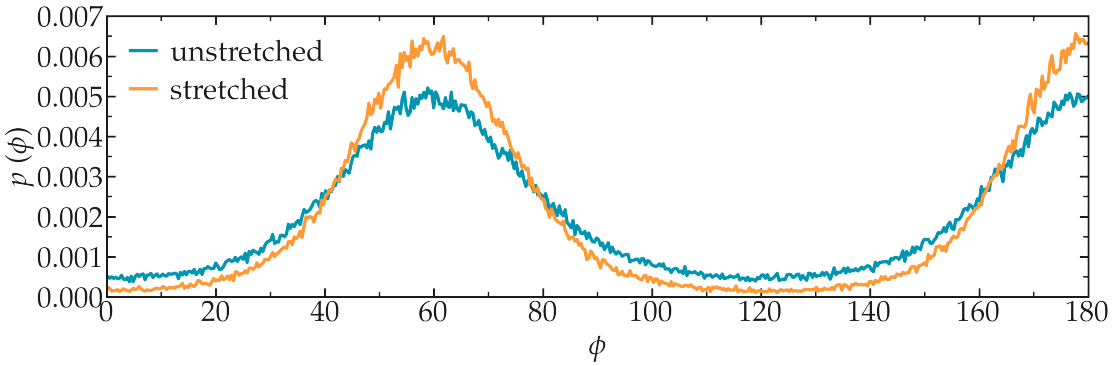
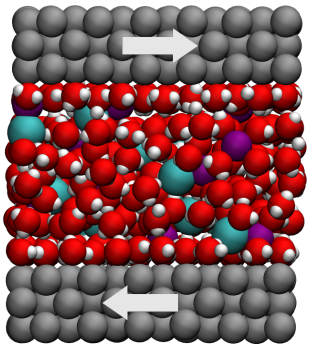


Figure: Probability distribution for the dihedral angle ϕ, for a streched and for an unstretched PEG molecule.

4

Nanosheared electrolyte

Aqueous NaCl solution sheared by two walls



The objective of this tutorial is to simulate an electrolyte nanoconfined and sheared by two walls. Some properties of the sheared fluid, such as its density and velocity profiles, will be extracted.

This tutorial illustrates some key aspects of combining a fluid and a solid in the same simulation. A major difference with Polymer in water is that here a rigid four points water model named TIP4P is used. TIP4P is one of the most common water model due to its high accuracy.

If you are completely new to LAMMPS, I recommend you to follow this tutorial on a simple Lennard Jones fluid first.

4.1 System preparation

The fluid and walls must first be generated, and then equilibrated at reasonable temperature and pressure.

4.1.1 System generation

Create a new folder called *systemcreation/*. Within *systemcreation/*, open a blank file called *input.lammps*, and copy the following lines into it:

```
units real
atom_style full
bond_style harmonic
angle_style harmonic
pair_style lj/cut/tip4p/long 1 2 1 1 0.1546 12.0
kspace_style pppm/tip4p 1.0e-4
```

These lines are used to define the most basic parameters, including the *atom*, *bond*, and *angle* styles, as well as interaction potential. Here *lj/cut/tip4p/long* imposes a Lennard Jones potential with a cut-off at 12 Å and a long range Coulomb potential.

So far, the commands are relatively similar to the previous tutorial (Polymer in water), with two major differences; the use of *lj/cut/tip4p/long* and *pppm/tip4p*, instead of *lj/cut/coul/long* and pppm. These two tip4p-specific commands allow us to model a four point water molecule without explicitly defining the fourth massless atom *M*. The value of 0.1546 Å corresponds to the *O-M* distance and is given by the water model. Here, TIP4P-2005 is used.

About lj/cut/tip4p/long pair style

The *lj/cut/tip4p/long* pair style is similar to the conventional Lennard Jones + Coulomb interaction, except that it is made specifically for four point water model (tip4p). The atom of the water model will be type 1 (O) and 2 (H). All the other atoms of the simulations are treated *normally* with long range coulomb interaction.

Let us create the box by adding the following lines to *input.lammps*:

```
lattice fcc 4.04
region box block -3 3 -3 3 -7 7
create_box 5 box &
bond/types 1 &
angle/types 1 &
extra/bond/per/atom 2 &
extra/angle/per/atom 1 &
extra/special/per/atom 2
```

The *lattice* command defines the unit cell. Here, the face-centered cubic (fcc) lattice with a scale factor of 4.04 has been chosen for the future positioning of the atoms of the walls.

The *region* command defines a geometric region of space. By choosing *xlo=-3* and *xhi=3*, and because we have previously chosen a lattice with scale factor of 4.04, the region box extends from -12.12 Å to 12.12 Å along the x direction.

The *create_box* command creates a simulation box with 5 types of atoms: the oxygen and hydrogen of the water molecules, the two ions (Na^+, Cl^-), and the atom of the walls. The *create_box* command extends over 6 lines thanks to the & character. The second and third lines are used to indicate that the simulation contains 1 type of bond and 1 type of angle (both required by the water molecule). The parameters for these bond and angle constraints will be given later. The three last lines are for memory allocation.

Now, we can add atoms to the system. First, let us create two sub-regions corresponding respectively to the two solid walls, and create a larger region from the union of the two regions. Then, let us create atoms of type 5 (the wall) within the two regions. Add the following lines to *input.lammps*:

```
region rbotwall block -3 3 -3 3 -4 -3
region rtopwall block -3 3 -3 3 3 4
region rwall union 2 rbotwall rtopwall
create_atoms 5 region rwall
```

Atoms will be placed at the positions of the previously defined lattice, thus forming fcc solids.

In order to add the water molecules, first download the molecule template and place it within *systemcreation/*. The template contains all the necessary information concerning the water molecule, such as atom positions, bonds, and angle.

Add the following lines to *input.lammps*:

```
region rliquid block INF INF INF INF -2 2
molecule h2omol RigidH2O.txt
create_atoms 0 region rliquid mol h2omol 482793
```

Withing the last four lines, a *region* named *rliquid* for depositing the water molecules is created based on the last defined lattice, which is *fcc 4.04*.

The *molecule* command opens up the molecule template named *RigidH2O.txt*, and names the associated molecule *h2omol*.

Molecules are created on the *fcc 4.04* lattice by the *create_atoms* command. The first parameter is '0', meaning that the atom ids from the *RigidH2O.txt* file will be used. The number *482793* is a seed that is required by LAMMPS, it can be any positive integer.

Finally, let us create 30 ions (15 Na^+ and 15 Cl^-) in between the water molecules, by adding the following commands to *input.lammps*:

```
create_atoms 3 random 15 52802 rliquid overlap 0.3 maxtry 500
create_atoms 4 random 15 90182 rliquid overlap 0.3 maxtry 500
set type 3 charge 1
set type 4 charge -1
```

Each *create_atoms* command will add 15 ions at random positions within the 'rliquid' region, ensuring that there is no *overlap* with existing molecules. Feel free to increase or decrease the salt concentration by changing the number of desired ions. To keep the system charge neutral, always insert the same number of Na^+ and Cl^-, unless there are other charges in the system.

The charges of the newly added ions are specified by the two *set* commands.

Before starting the simulation, we still need to define the parameters of the simulation: the mass of the 5 atom types (O, H, Na^+, Cl^-, and wall), the pairwise interaction parameters (here the parameters for the Lennard-Jones potential), and the bond and angle parameters. Copy the following line into input.lammps:

```
include ../PARM.lammps
include ../GROUP.lammps
```

Create a new text file, call it *PARM.lammps*, and copy it next to the *systemcreation/* folder. Copy the following lines into PARM.lammps:

```
mass 1 15.9994 # water
mass 2 1.008 # water
mass 3 28.990 # ion
mass 4 35.453 # ion
mass 5 26.9815 # wall

pair_coeff 1 1 0.185199 3.1589 # water
pair_coeff 2 2 0.0 0.0 # water
pair_coeff 3 3 0.04690 2.4299 # ion
pair_coeff 4 4 0.1500 4.04470 # ion
pair_coeff 5 5 11.697 2.574 # wall
pair_coeff 1 5 0.4 2.86645 # water-wall

bond_coeff 1 0 0.9572 # water

angle_coeff 1 0 104.52 # water
```

Each *mass* command assigns a mass in grams/mole to an atom type. Each *pair_coeff* assigns respectively the depth of the LJ potential (in Kcal/mole), and the distance (in Ångstrom) at which the particle-particle potential energy is 0.

About the parameters

The parameters for water correspond to the TIP4P/2005 water model, and the parameters for Na^+ and Cl^- are taken from the CHARMM-27 force field.

As already seen in previous tutorials, and with the important exception of *pair_coeff 1 5*, only pairwise interaction between atoms of identical type were assigned. By default, LAMMPS calculates the pair coefficients for the interactions between atoms of different types (i and j) by using geometrical average: $\epsilon_{ij} = (\epsilon_{ii} + \epsilon_{jj})/2$, $\sigma_{ij} = (\sigma_{ii} + \sigma_{jj})/2$. Other rules for cross coefficients can be set with the *pair_modify* command, but for the sake of simplicity, let us keep the default option here.

By default, the value of $\epsilon_{1\text{-}5} = 5.941$ kcal/mol would be extremely high (compare to the water-water energy $\epsilon_{1\text{-}1} = 0.185199$ kcal/mol), which would make the surface extremely hydrophilic. The walls were made less hydrophilic by reducing the LJ energy of interaction $\epsilon_{1\text{-}5}$ to a lower value.

The *bond_coeff*, which is here used for the O-H bond of the water molecule, sets both the energy of the harmonic potential and the equilibrium distance in Ångstrom. The value is *0* for the energy, because we are going to use a rigid model for the water molecule. The shape of the molecule will be preserved later by the *shake* algorithm. Similarly, the angle coefficient here for the H-O-H angle of the water molecule sets the energy of the harmonic potential (also 0) and the equilibrium angle is in degree.

Let us also create another file called *GROUP.lammps* next to *PARM.lammps*, and copy the following lines into it:

```
group H2O type 1 2
group Na type 3
group Cl type 4
group ions union Na Cl
group fluid union H2O ions

group wall type 5
region rtop block INF INF INF INF 0 INF
region rbot block INF INF INF INF INF 0
group top region rtop
group bot region rbot
group walltop intersect wall top
group wallbot intersect wall bot
```

To avoid high density and pressure, let us add the following lines to *input.lammps* to delete a few of the water molecules:

```
delete_atoms random fraction 0.15 yes H2O NULL 482793 mol yes
```

Finally, add the following lines to *input.lammps*:

```
run 0

write_data system.data
write_dump all atom dump.lammpstrj
```

With *run 0*, the simulation will run for 0 step, which is enough for creating the system and saving the final state.

The *write_data* creates a file named *system.data* containing all the information required to restart the simulation from the final configuration generated by this input file.

The *write_dump* command prints the final positions of the atoms, and can be opened with VMD to visualize the system.

Run the *input.lammps* file using LAMMPS.

Always check that your system has been correctly created by looking at the periodic images. Atomic defects may occur at the boundary.

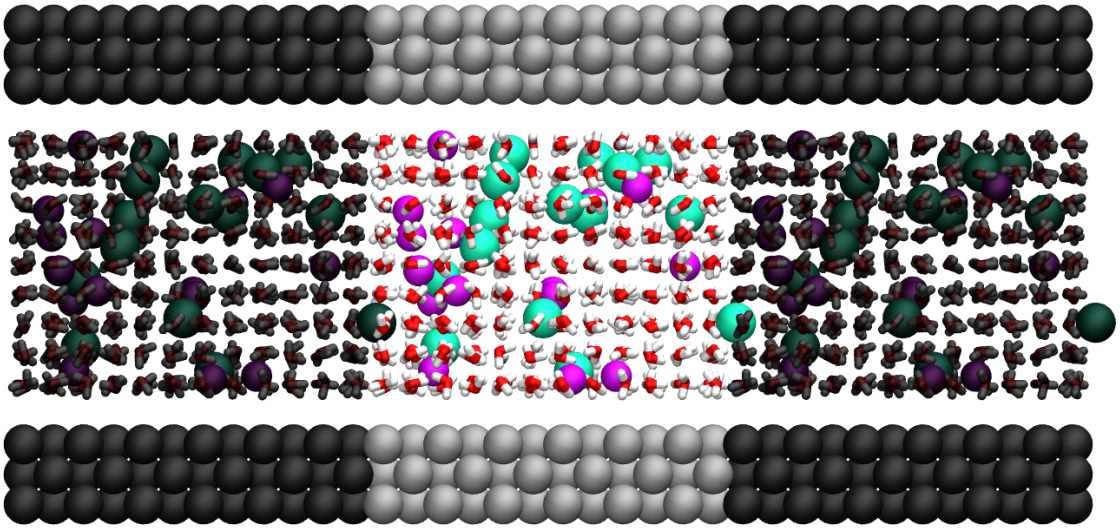


Figure: Side view of the system. Periodic images are represented in darker color. Water molecules are in red and white, Na^+ ions in purple, Cl^- ions in lime, and wall atoms in gray. Note the absence of atomic defect at the cell boundaries. See the corresponding video.

4.1.2 Energy minimization

Why is energy minimization necessary?

It is clear from the way the system has been created that the atoms are not at equilibrium distances from each others. Indeed, some of the ions added using the *create_atoms* commands are too close to the water molecules. If we were to start a *normal* (i.e. with a timestep of about 1 fs) molecular dynamics simulation now, the atoms would exert huge forces on each others, accelerate brutally, and the simulation would likely fail.

Dealing with overlapping atoms

MD simulations failing due to overlapping atoms are extremely common. If it occurs, you can either

- delete the overlapping atoms using the *delete_atoms* command of LAMMPS,
- move the atoms to more reasonable distances before the simulation starts using energy minimization, or using molecular dynamics with a small timestep.

Let us move the atoms and place them in more energetically favorable positions before starting the simulation. Let us call this step *energy minimization*, although it is not a conventional *minimization* as done for instance in tutorial Lennard Jones fluid.

To perform this energy minimization, let us create a new folder named *minimization/* next to *systemcreation/*, and create a new input file named *input.lammps* in it. Copy the following lines in *input.lammps*:

```
boundary p p p
units real
atom_style full
bond_style harmonic
angle_style harmonic
pair_style lj/cut/tip4p/long 1 2 1 1 0.1546 12.0
kspace_style pppm/tip4p 1.0e-4

read_data ../systemcreation/system.data

include ../PARM.lammps
include ../GROUP.lammps
```

The only difference with the previous input is that, instead of creating a new box and new atoms, we open the previously created file *system.data* located in *systemcreation/*. The file *system.data* contains the definition of the simulation box and the positions of the atoms.

Now, let us create a first simulation step using a relatively small timestep (0.5 fs), as well as a low temperature of $T = 1$ K:

```
fix mynve fluid nve/limit 0.1
fix myber fluid temp/berendsen 1 1 100
fix myshk H2O shake 1.0e-4 200 0 b 1 a 1
timestep 0.5
```

Just like *fix nve*, the fix *nve/limit* performs constant NVE integration to update positions and velocities of the atoms at each timestep, but also limits the maximum distance atoms can travel at each timestep. Here, only the fluid molecules and ions will move.

The *fix temp/berendsen* rescales the velocities of the atoms to force the temperature of the system to reach the desired value of 1 K, and the shake algorithm is used in order to maintain the shape of the water molecules.

Let us also print the atom positions in a *.lammpstrj* file by adding the following line to *input.lammps*:

```
dump mydmp all atom 1000 dump.lammpstrj
thermo 200
```

Finally, let us run for 4000 steps. Add the following lines into *input.lammps*:

```
run 4000
```

In order to better equilibrate the system, let us perform two additional steps with a larger timestep and a larger imposed temperature:

```
fix myber fluid temp/berendsen 300 300 100
timestep 1.0

run 4000

unfix mynve
fix mynve fluid nve

run 4000

write_data system.data
```

For the last of the 3 steps, fix *nve* is used instead of *nve/limit*, which will allow for a better relaxation of the atom positions.

When running the *input.lammps* file with LAMMPS, you should see that the total energy of the system decreases during the first of the 3 steps, before re-increasing a little after the temperature is increased from 1 to 300 K.

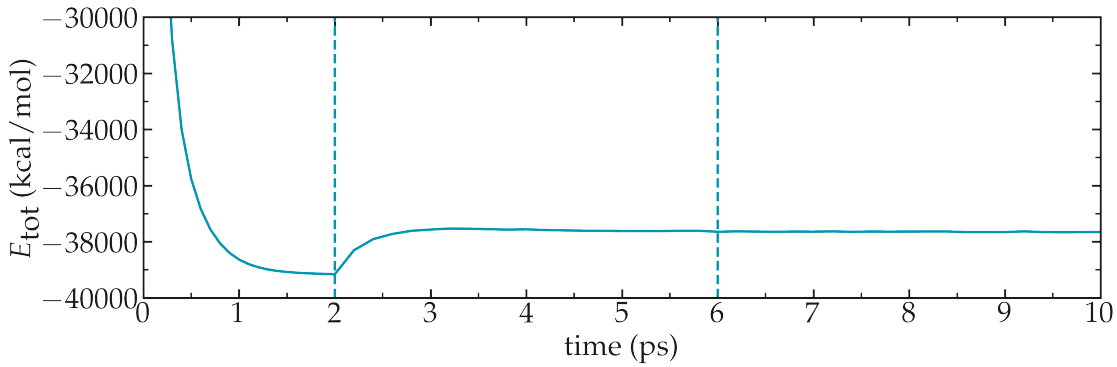


Figure: Energy as a function of time extracted from the log file using *Python* and *lammps_logfile*.

If you look at the trajectory using VMD, you will see some of the atoms, in particular the one that where initially in problematic positions.

4.1.3 System equilibration

Now, let us equilibrate further the entire system by letting both fluid and piston relax at ambient temperature.

Create a new folder called *equilibration/* next to the previously created folders, and create a new *input.lammps* file in it. Add the following lines into *input.lammps*:

```
boundary p p p
units real
atom_style full
bond_style harmonic
angle_style harmonic
pair_style lj/cut/tip4p/long 1 2 1 1 0.1546 12.0
kspace_style pppm/tip4p 1.0e-4

read_data ../minimization/system.data

include ../PARM.lammps
include ../GROUP.lammps
```

Finally, let us complete the *input.lammps* file:

```
fix mynve all nve
fix myber all temp/berendsen 300 300 100
fix myshk H2O shake 1.0e-4 200 0 b 1 a 1
fix myrct all recenter NULL NULL 0
timestep 1.0
```

The fix *recenter* has no influence on the dynamics, but will keep the system in the center of the box, which makes the visualization easier.

Then, add the following lines to *input.lammps* for the trajectory visualization and output:

```
dump mydmp all atom 1000 dump.lammpstrj
thermo 500
variable walltopz equal xcm(walltop,z)
variable wallbotz equal xcm(wallbot,z)
variable deltaz equal v_walltopz-v_wallbotz
fix myat1 all ave/time 100 1 100 v_deltaz file interwall_distance.dat
```

The first two variables extract the centers of mass of the two walls. Then, the *deltaz* variable is used to calculate the distance between the two variables *walltopz* and *wallbotz*, i.e. the distance between the two walls.

Finally, let us add the *run* command:

```
run 30000
write_data system.data
```

Run the *input.lammps* file using LAMMPS.

As seen from the data printed by *fix myat1*, the distance δ_z between the two walls reduces until it reaches an equilibrium value.

Note that it is generaly recommended to run longer equilibration. Here for instance, the slowest process in the system is probably the ionic diffusion. Therefore the equilibration should in principle be longer than the time the ions need to diffuse over the size of the pore ($\approx 1.2\,\text{nm}$), i.e. of the order of half a nanosecond.

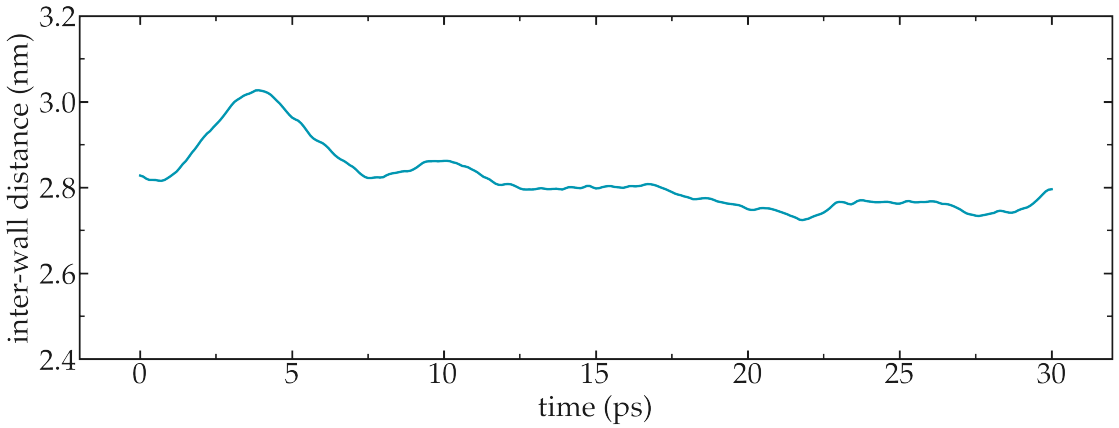


Figure: Distance between the walls as a function of time. After a few pico seconds, the distance between the two walls equilibrates near its final value.

4.2 Imposed shearing

From the equilibrated configuration, let us impose a laterial motion to the two walls and shear the electrolyte. In a new folder called *shearing/*, create a new *input.lammps* file that starts like the previous ones:

```
boundary p p p
units real
atom_style full
bond_style harmonic
angle_style harmonic
pair_style lj/cut/tip4p/long 1 2 1 1 0.1546 12.0
kspace_style pppm/tip4p 1.0e-4
```

Let us import the previously equilibrated data, include the parameter and group files, and then deal with the dynamics of the system.

```
read_data ../equilibration/system.data

include ../PARM.lammps
include ../GROUP.lammps

fix mynve all nve
compute Tfluid fluid temp/partial 0 1 1
fix myber1 fluid temp/berendsen 300 300 100
fix_modify myber1 temp Tfluid
compute Twall wall temp/partial 0 1 1
fix myber2 wall temp/berendsen 300 300 100
fix_modify myber2 temp Twall
fix myshk H2O shake 1.0e-4 200 0 b 1 a 1
fix myrct all recenter NULL NULL 0
```

One difference here is that two thermostats are used, one for the fluid (*myber1*) and one for the solid (*myber2*). The use of *fix_modify* together with *compute* ensures that the right temperature value is used by the thermostats.

The use of temperature *compute* with *temp/partial 0 1 1* is meant to exclude the x coordinate from the thermalisation, which is important since a large velocity will be imposed along x.

Then, let us impose the velocity of the two walls by adding the following command to *input.lammps*:

```
fix mysf1 walltop setforce 0 NULL NULL
fix mysf2 wallbot setforce 0 NULL NULL
velocity wallbot set -2e-4 NULL NULL
velocity walltop set 2e-4 NULL NULL
```

The *setforce* commands cancel the forces on *walltop* and *wallbot*, respectively. Therefore the atoms of the two groups do not experience any force from the rest of the system. In absence of force acting on those atoms, they will conserve their initial velocity.

The *velocity* commands act only once and impose the velocity of the atoms of the groups *wallbot* and *walltop*, respectively.

Finally, let us dump the atom positions, extract the velocity profiles using several *ave/chunk* commands, extract the force applied on the walls, and then run for 200 ps Add the following lines to *input.lammps*:

```
dump mydmp all atom 5000 dump.lammpstrj
thermo 500
thermo_modify temp Tfluid

compute cc1 H2O chunk/atom bin/1d z 0.0 1.0
compute cc2 wall chunk/atom bin/1d z 0.0 1.0
compute cc3 ions chunk/atom bin/1d z 0.0 1.0

fix myac1 H2O ave/chunk 10 15000 200000 &
cc1 density/mass vx file water.profile_1A.dat
fix myac2 wall ave/chunk 10 15000 200000 &
cc2 density/mass vx file wall.profile_1A.dat
fix myac3 ions ave/chunk 10 15000 200000 &
cc3 density/mass vx file ions.profile_1A.dat

fix myat1 all ave/time 10 100 1000 f_mysf1[1] f_mysf2[1] file forces.dat

timestep 1.0
run 200000
write_data system.data
```

Here, a binning of 1 Å is used. For smoother profiles, you can reduce its value.

The averaged velocity profile of the fluid can be plotted. As expected here, the velocity of the fluid is found to increase linearly along z.

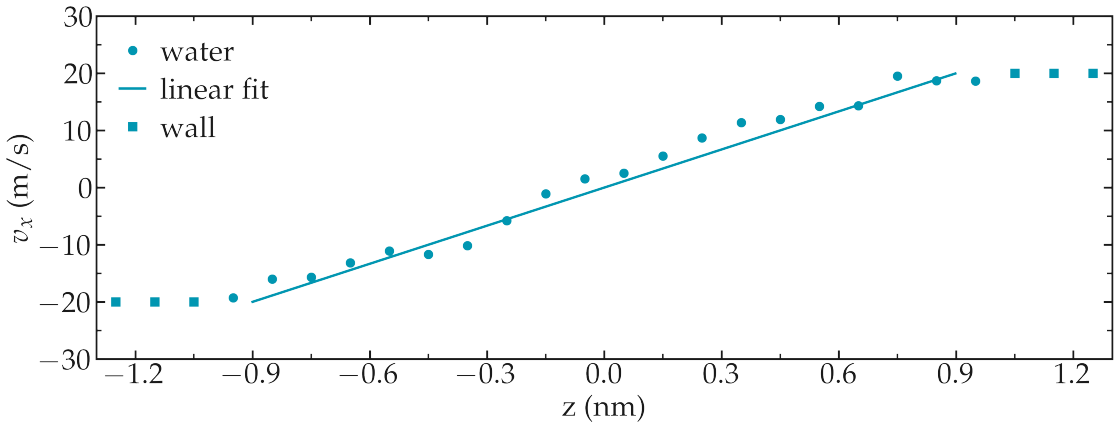


Figure: Velocity profiles for water molecules, ions and walls along the z axis. The line is a linear fit assuming that the pore size is $h = 1.8\,\text{nm}$.

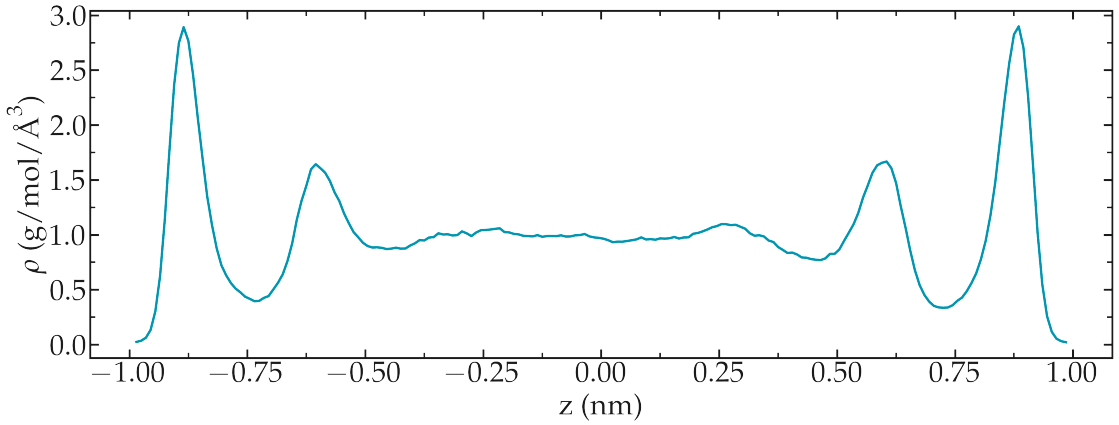


Figure: Water density ρ profile along the z axis.

From the force applied by the fluid on the solid, one can extract the stress within the fluid, which allows one to measure its viscosity $\dot{\eta}$ according to gravelle2021: $\eta = \tau/\dot{\gamma}$ where τ is the stress applied by the fluid on the shearing wall, and $\dot{\gamma}$ the shear rate (which is imposed here). Here the shear rate is approximatively $\dot{\gamma} = 16 \cdot 10^9\,\text{s}^{-1}$, and using a surface area of $A = 6 \cdot 10^{-18}\,\text{m}^2$, one gets an estimate for the shear viscosity for the confined fluid of $\eta = 6.6\,\text{mPa.s}$

The viscosity calculated at such high shear rate may differ from the expected *bulk* value. In general, it is recommanded to use a lower value for the shear rate. Note that for lower shear rate, the ratio noise-to-signal is larger, and longer simulations are needed.

Another important point is that the viscosity of a fluid next to a solid surface is typically larger than in bulk due to interaction with the walls. Therefore, one expects the present simulation to return a viscosity that is slightly larger than what would be measured in absence of wall.

You can access all the input scripts and data files that are used in these tutorials from the inputs folder on Github. This repository also contains the full solutions to the exercises.

4.3 Going further with exercises

Each exercise comes with a proposed solution, see Solutions to the exercises.

4.3.1 Induce a Poiseuille flow

Instead of inducing a shearing of the fluid using the walls, induce a net flux of the liquid in the direction tangential to the walls. The walls must be kept immobile.

Extract the velocity profile, and make sure that the resulting velocity profile is consistent with the Poiseuille equation, which can be derived from the Stokes equation $\eta\nabla\mathbf{v} = -\mathbf{f}\rho$ where f is the applied force, ρ is the fluid density, η is the fluid viscosity.

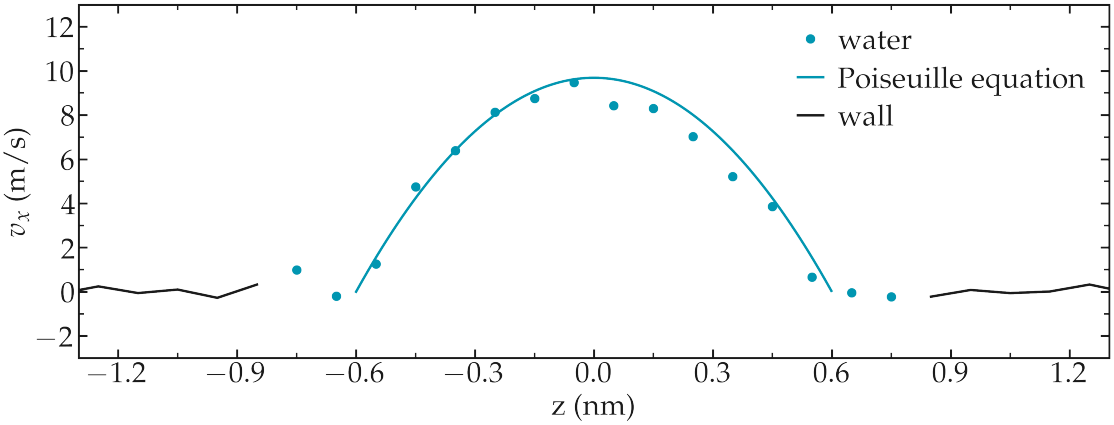


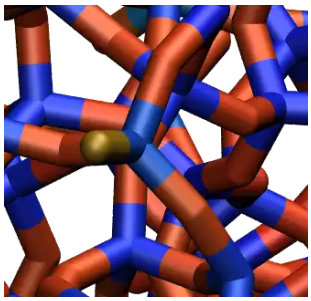
Figure: Velocity profiles of the water molecules along the z axis (disks). The line is the Poiseuille equation.

An important step is to choose the proper value for the addidional force.

5

Reactive silicon dioxide

Simulating a chemically reactive structure



The objective of this tutorial is to use the reactive force field named *reaxff*. This force field allows for the calculation of chemical bond formation.

The system simulated here is a block of silicon dioxide (SiO2), that is deformed until rupture. A particular attention is given to the evolution of the charges of the atoms during the deformation of the structure, and the chemical reactions occurring due to the deformation are tracked and discussed.

If you are completely new to LAMMPS, I recommend you to follow this tutorial on a simple Lennard Jones fluid first.

5.1 Prepare and relax

Create a folder, name it *RelaxSilica/*, and download the initial topology of a small amorphous silica structure.

The system was created by temperature annealing using another force field named vashishta.

In case you are interested in the input creation, the input file used for creating the initial topology is available here.

If you open the *silica.data* file, you can see by looking at the Atoms section that all silicon atoms have the same charge $q = 1.1\,\mathrm{e}$, and all oxygen atoms the charge $q = -0.55\,\mathrm{e}$. This is common with classical force field, and will change once reaxff is used. Let us keep that in mind for now.

The first step we need to perform here is to relax the structure with *reaxff*, which we are gonna do using molecular dynamics. To make sure that the system equilibrates nicely, let us track some changes over time.

Create an input file called *input.lammps* in *RelaxSilica/*, and copy the following lines in it:

```
units real
atom_style full

read_data silica.data

mass 1 28.0855 # Si
mass 2 15.999 # O
```

So far, the input is very similar to what was seen in the previous tutorials. Some basic parameters are defined (*units*, *atom_style* and *masses*), and the *.data* file is imported by the *read_data* command. Now let us enter 3 crucial lines in the *input.lammps* file:

```
pair_style reaxff NULL safezone 3.0 mincap 150
pair_coeff * * reaxCHOFe.ff Si O
fix myqeq all qeq/reaxff 1 0.0 10.0 1.0e-6 reaxff maxiter 400
```

Here, the reaxff *pair_style* is used with no control file. The *safezone* and *mincap* keywords have been added to avoid memory allocation issue, which sometimes can trigger the segmentation faults and bondchk failed errors.

The *pair_coeff* uses the reaxCHOFe.ff file which is assumed to be saved within *RelaxSilica/*. For consistency, the atoms of type 1 are set as silicon (Si), and the atoms of type 2 as oxygen (O).

Finally, the *fix qeq/reaxff* is used to perform charge equilibration. The charge equilibration occurs at every step. The values 0 and 10.0 are the low and the high cutoffs, respectively, and $1.0e-6$ is a tolerance. Finally, *maxiter* sets a upper limit to the number of attempt to equilibrate the charge.

Note

If the charge does not properly equilibrate despite the 400 attempts, a warning will appear. Such warnings are likely to appear at the beginning of the simulation if the initial charges are too far from the equilibrium values.

Then, let us add some commands to the *input.lammps* file to measure the evolution of the charges during the simulation:

```
group grpSi type 1
group grpO type 2
variable totqSi equal charge(grpSi)
variable totqO equal charge(grpO)
variable nSi equal count(grpSi)
variable nO equal count(grpO)
variable qSi equal v_totqSi/${nSi}
variable qO equal v_totqO/${nO}
```

Let us also print the charge in the *.log* file by using *thermo_style*, and create a *.lammpstrj* file for visualization. Add the following lines to the *input.lammps*:

```
dump dmp all custom 100 dump.lammpstrj id type q x y z
thermo 5
thermo_style custom step temp etotal press vol v_qSi v_qO
```

Let us also use the *fix reaxff/species* to evaluate what species are present within the simulation. It will be useful later, when the system is deformed:

```
fix myspec all reaxff/species 5 1 5 species.log element Si O
```

Here, the information will be printed every 5 steps in a file named *species.log*.

Let us perform a very short run using anisotropic NPT command and relax the density of the system.

```
velocity all create 300.0 3482028
fix mynpt all npt temp 300.0 300.0 100 aniso 1.0 1.0 1000
timestep 0.5

run 5000
```

Run the *input.lammps* file using LAMMPS. As can be seen from *species.log*, only one species is detected, called *Si192O384*, which is the entire system.

As the simulation progresses, you can see that the charges of the atoms are fluctuating since the charge of every individual atom is adjusting to its local environnement.

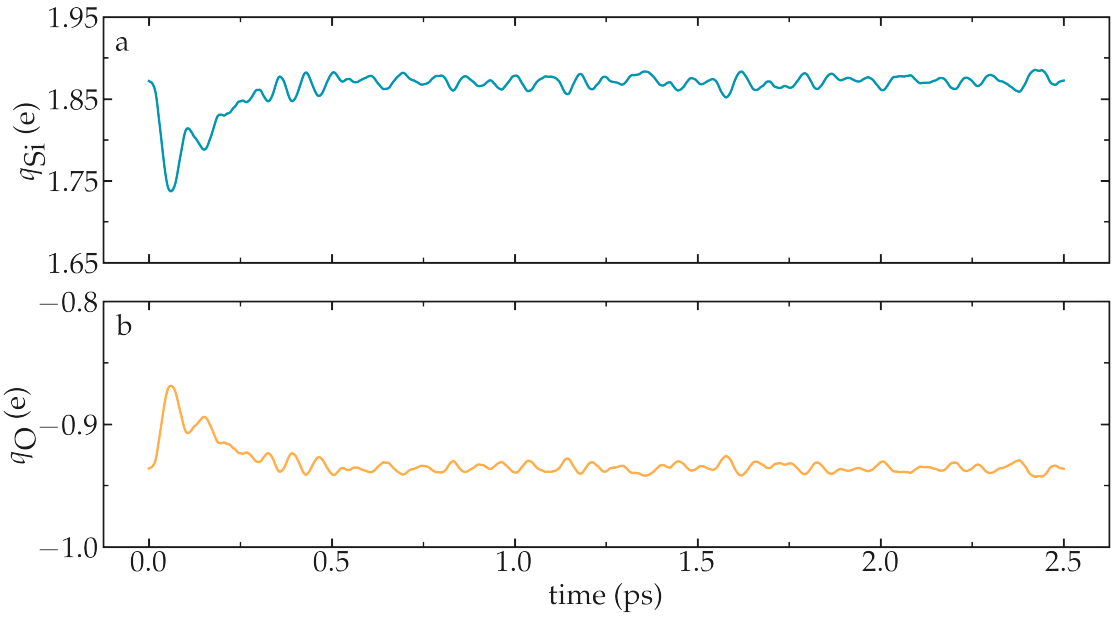


Figure: Average charge per atom of the silicon (a) and oxygen (b) atoms during equilibration.

One can see that the charges of the atoms are strongly fluctuating at the beginning of the simulation. This early fluctuation correlates with a rapid volume change of the box, during which one can guess that the inter atomic distances are also quickly changing.

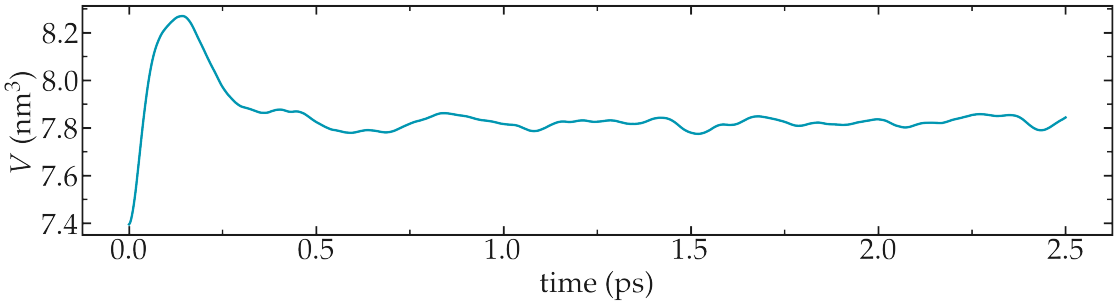


Figure: Volume of the system as a function of time.

Since each atom has a charge that depends on its local environnement, the charge values are expected to be different for every atom in the system. We can plot the charge distribution $P(q)$, using the charge values printed in the *.lammptrj* file.

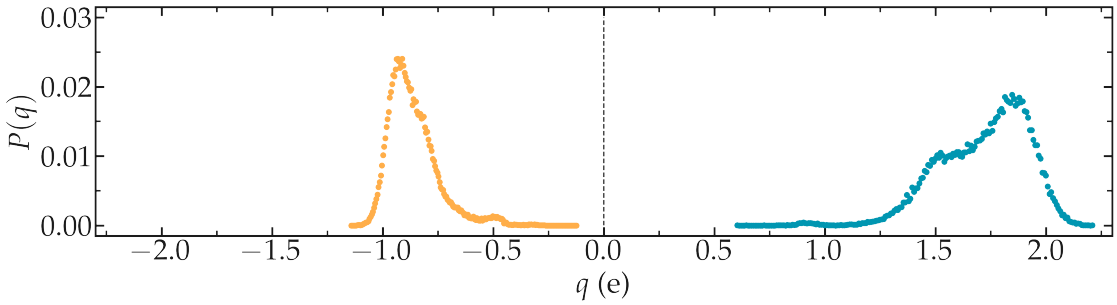


Figure: Probability distribution of charge of silicon (positive, blue) and oxygen (negative, orange) atoms during equilibration.

Using VMD and coloring the atoms by their charges, one can see that the atoms with the extreme-most charges are located at defects in the amorphous structure (here at the positions of the dandling oxygen groups).

5.2 Deform the structure

Let us apply a deformation to the structure in order to force some Si − O bonds to break and/or re-assemble.

Next to *RelaxSilica/*, create a folder, call it *Deform/* and create a file named *input.lammps* in it. Copy the same lines as previously in input.lammps:

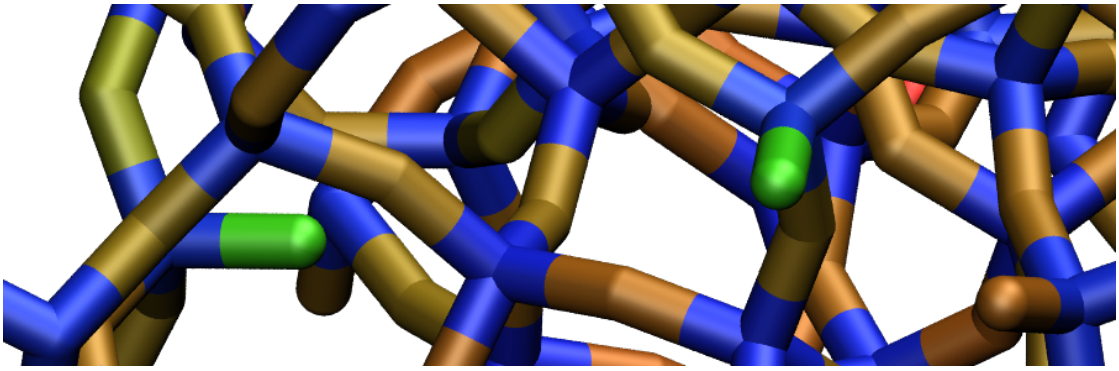


Figure: Amorphous silica colored by charges using VMD. Dandling oxygen groups appear in green. To color the atoms by their charge in VMD, use *Charge* as coloring method in the representation windows, and then tune the *Color scale* in the *Color control windows*.

```
units real
atom_style full

read_data ../RelaxSilica/silica-relaxed.data

mass 1 28.0855 # Si
mass 2 15.999 # O

pair_style reaxff NULL safezone 3.0 mincap 150
pair_coeff * * ../RelaxSilica/reaxCHOFe.ff Si O
fix myqeq all qeq/reaxff 1 0.0 10.0 1.0e-6 reaxff maxiter 400
```

The only differences with the previous *input.lammps* file are the paths to the *.data* and *.ff* files located within *RelaxSilica/*. Copy the following lines as well:

```
group grpSi type 1
group grpO type 2
variable totqSi equal charge(grpSi)
variable totqO equal charge(grpO)
variable nSi equal count(grpSi)
variable nO equal count(grpO)
variable qSi equal v_totqSi/${nSi}
variable qO equal v_totqO/${nO}

dump dmp all custom 100 dump.lammpstrj id type q x y z
thermo 5
thermo_style custom step temp etotal press vol v_qSi v_qO
fix myspec all reaxff/species 5 1 5 species.log element Si O
```

Then, let us use *fix nvt* instead of *fix npt* to apply a thermostat, but no barostat because the box deformations will be imposed.

```
fix mynvt all nvt temp 300.0 300.0 100
timestep 0.5
```

Then, let us use run for 5000 steps, then apply the *fix deform* for elongating progressively the box along *x*. Let us apply fix deform during 25000 steps. Add the following line to *input.lammps*:

```
run 5000

fix mydef all deform 1 x erate 5e-5

run 25000

write_data silica-deformed.data
```

During the deformation, the charges progressively change until the structure eventually breaks down. After the structure breaks down, the charges equilibrate near new average values that differ from the starting averages. The difference between the initial and the final charges can be explained by presence of a new solid/vacuum interface: surface atoms typically have different charges compared to bulk atoms.

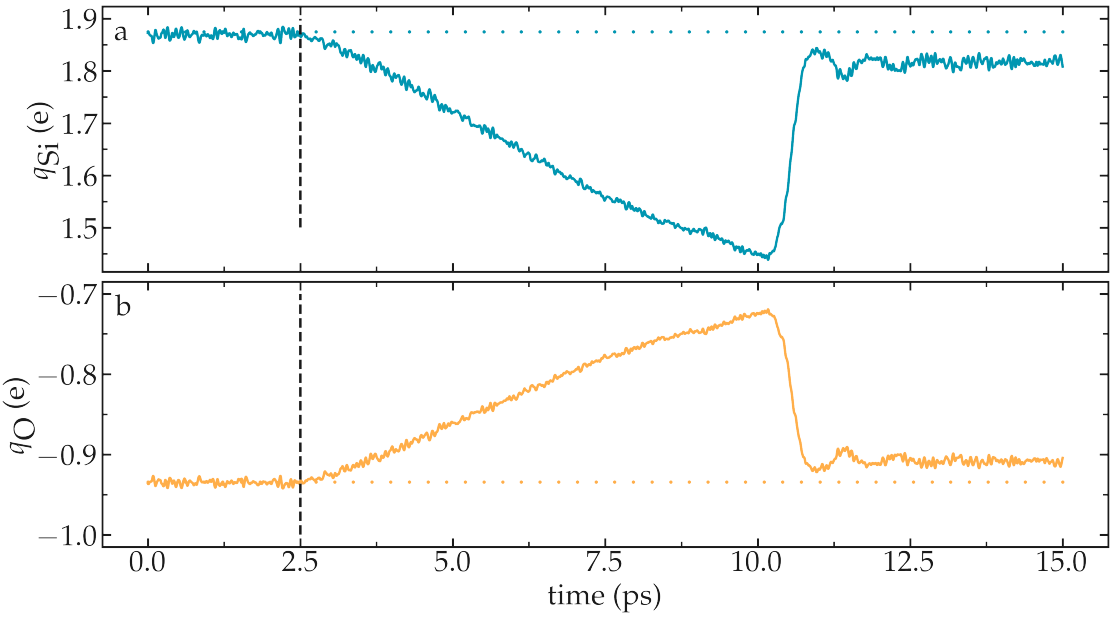


Figure: Average charge per atom of the silicon (a) and oxygen (b). The vertical dashed lines marks the beginning of the deformation.

There is also a strong increase in temperature during the rupture of the material.

At the end of the deformation, one can visualize the broken material using VMD. Notice the different charge of the atoms located near the interface, compared to the atoms located in the bulk of the material.

One can have a look at the charge distribution after deformation, as well as during the deformation.

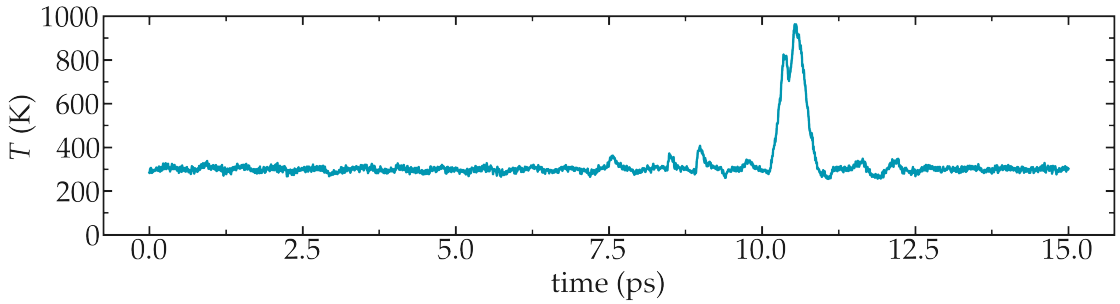


Figure: Temperature of the system over time.

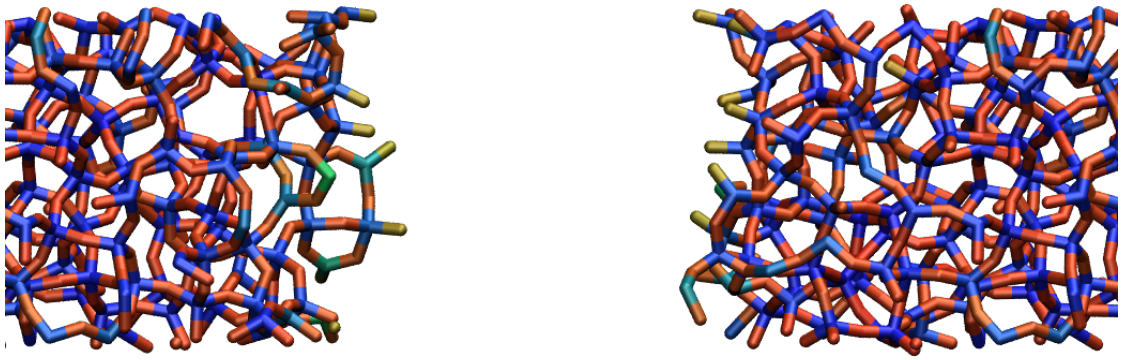


Figure: Amorphous silicon oxide after deformation. The atoms are colored by charges using VMD.

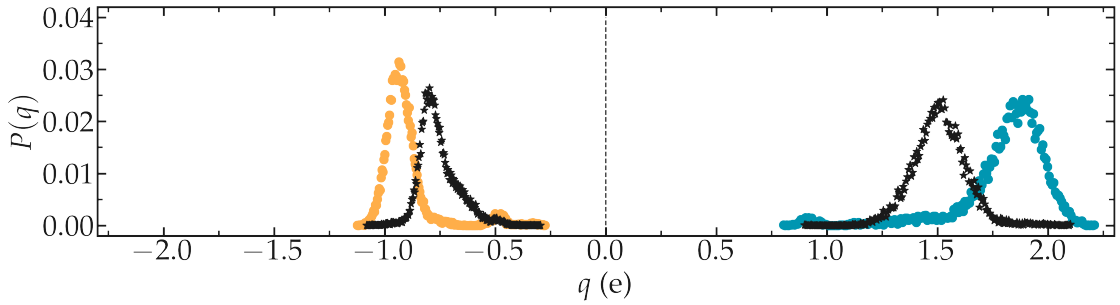


Figure: Distribution of charge of silicon (positive, blue) and oxygen (negative, orange) after deformation. The stars correspond to the charge distribution during deformation.

The final charge distribution slightly differs from the previously calculated, which was to be expected as the material was broken. In my case, no new species was formed during the simulation, as can be seen from the *species.log* file:

```
#  Timestep    No_Moles    No_Specs   Si1920384
       5           1           1           1
(...)
#  Timestep    No_Moles    No_Specs   Si1920384
    30000           1           1           1
```

Sometimes, O_2 molecules are formed during the deformation. If this is the case, the *species.log* file will look like:

```
#  Timestep    No_Moles    No_Specs   Si1920384
          5           1           1           1
(...)
#  Timestep    No_Moles    No_Specs   Si1920382          O2
      30000           1           1           1           1
```

You can access all the input scripts and data files that are used in these tutorials from the inputs folder on Github. This repository also contains the full solutions to the exercises.

5.3 Going further with exercises

Each exercise comes with a proposed solution, see Solutions to the exercises.

5.3.1 Add O2 molecules

Add O_2 molecules to the previously equilibrated structure. Equilibrate it again, and extract the charge density profile along the x axis.

Here, the O_2 molecule is simply made of 2 oxygen atoms that are not connected by any bond.

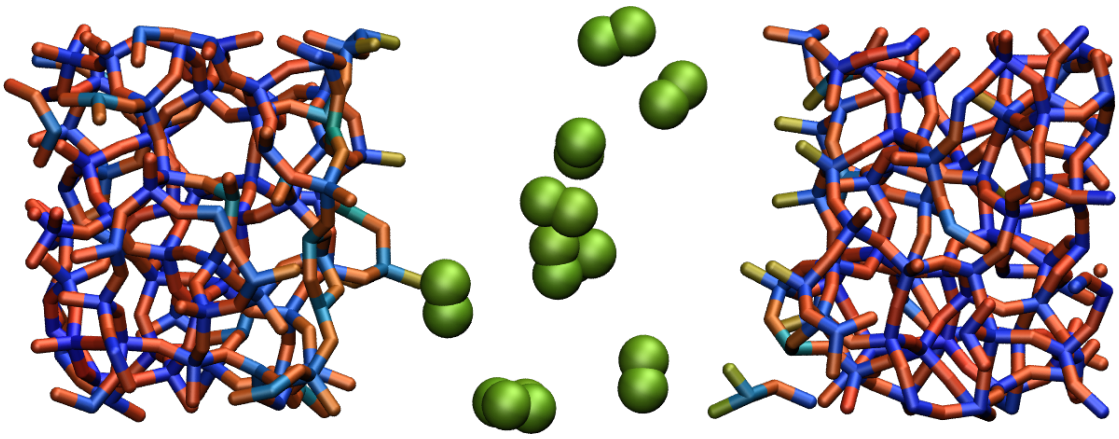


Figure: Deformed structure with some O_2 molecules

5.3.2 Decorate dandling oxygens

Under ambient conditions, dandling oxygen are typically terminated by hydrogen atoms. Improve the current structure by decorating some of the dandling oxygen atoms with hydrogen atoms.

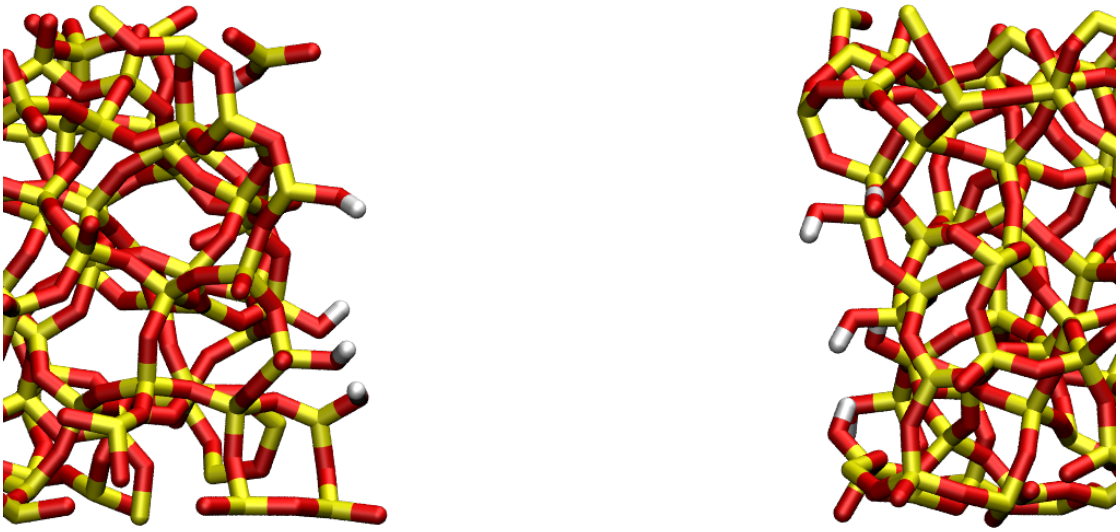
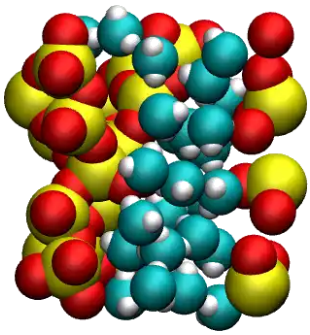


Figure: Hydrogen atoms are in white, oxygen in red, and silicon in yellow.

6

Water adsorption in silica

Dealing with a varying number of molecules



The objective of this tutorial is to combine molecular dynamics and grand canonical Monte Carlo simulations to compute the adsorption of water molecules in a cracked silica material.

This tutorial illustrates the use of the grand canonical ensemble in molecular simulation, an open ensemble in which the number of atoms within the simulation box is not constant. When using the grand canonical ensemble, it is possible to impose the chemical potential (or pressure, or fugacity) of a given fluid in a nanoporous structure.

If you are completely new to LAMMPS, I recommend you to follow this tutorial on a simple Lennard Jones fluid first.

6.1 Generation of the silica block

Let us first generate a block of amorphous silica (SiO2). To do so, we are going to replicate a building block containing 3 Si and 6 O atoms.

Create two folders side by side, and name them respectively *Potential/* and *SilicaBlock/*.

An initial data file for the SiO atoms can be downloaded by clicking here. Save it in *SilicaBlock/*. This data file contains the coordinates of the 9 atoms, their masses, and their charges. The *.data* file can be directly read by LAMMPS using the *read_data* command. Let us replicate these atoms using LAMMPS, and apply an annealing procedure to obtain a block of amorphous silica.

About annealing procedure

The annealing procedure consists of adjusting the system temperature in successive steps. Here, a large initial temperature is chosen to ensure the melting of the SiO2 structure. Then, several steps are used to progressively cool down the system until it solidifies and forms amorphous silica. Depending on the material, different cooling velocities can sometimes lead to different crystal structures or different degrees of defect.

6.1.1 Vashishta potential

Create a new input file named *input.lammps* in the *SilicaBlock/* folder, and copy the following lines in it:

```
units metal
boundary p p p
atom_style full
pair_style vashishta
neighbor 1.0 bin
neigh_modify delay 1
```

The main difference with some of the previous tutorials is the use of the *Vashishta* pair style. Download the *Vashishta* potential by clicking here, and copy it within the *Potential/* folder.

About the Vashishta potential

The Vashishta potential is a bond-angle energy based potential, it deduces the bonds between atoms from their relative positions. Therefore, there is no need to provide bond and angle information as we do with classic force fields like GROMOS or AMBER. When used with LAMMPS, the *Vashishta* potential requires the use of metal units system. Bond-angle energy based potentials are more computationally heavy than classical force fields and require the use of a smaller timestep, but they allow for the modelling of bond formation and breaking, which is what we need here as we want to create a crack in the silica.

Let us then import the system made of 9 atoms, replicate it four times in all three directions of space, thus creating a system with 576 atoms. Add the following lines to *input.lammps*:

```
read_data SiO.data
replicate 4 4 4
```

Then, let us specify the pair coefficients by indicating that the first atom type is *Si*, and the second is *O*. Let us also add a dump command for printing out the positions of the atoms every 5000 steps:

```
pair_coeff * * ../Potential/SiO.1990.vashishta Si O
```

Let us add some commands to *input.lammps* to help us follow the evolution of the system, such as its temperature, volume, and potential-energy:

```
dump dmp all atom 5000 dump.lammpstrj
variable myvol equal vol
variable mylx equal lx
variable myly equal ly
variable mylz equal lz
variable mypot equal pe
variable mytemp equal temp
fix myat1 all ave/time 10 100 1000 v_mytemp file temperature.dat
fix myat2 all ave/time 10 100 1000 &
v_myvol v_mylx v_myly v_mylz file dimensions.dat
fix myat3 all ave/time 10 100 1000 v_mypot file potential-energy.dat
thermo 1000
```

6.1.2 Annealing procedure

Finally, let us create the last part of our script. The annealing procedure is made of four consecutive runs. First, a 50 ps phase at $T = 6000$ K and isotropic pressure coupling with desired pressure $p = 100$ atm:

```
velocity all create 6000 4928459 rot yes dist gaussian
fix npt1 all npt temp 6000 6000 0.1 iso 100 100 1
timestep 0.001
run 50000
```

Then, a second phase during which the system is cooled down from $T = 6000$ K to $T = 4000$ K. An anisotropic pressure coupling is used, allowing all three dimensions of the box to evolve independently from one another:

```
fix npt1 all npt temp 6000 4000 0.1 aniso 100 100 1
run 50000
```

Then, let us cool down the system further while also reducing the pressure, then perform a small equilibration step at the final desired condition, $T = 300$ K and $p = 1$ atm.

```
fix npt1 all npt temp 4000 300 0.1 aniso 100 1 1
run 200000
fix npt1 all npt temp 300 300 0.1 aniso 1 1 1
run 50000

write_data amorphousSiO.data
```

Disclaimer – I created this procedure by intuition and not from proper calibration, do not copy it without making your own tests if you intend to publish your results.

Anisotropic versus isotropic barostat

Here, an isotropic barostat is used for the melted phase at $T = 6000\,\mathrm{K}$, and then an anisotropic barostat is used for all following phases. With the anisotropic barostat, all three directions of space are adjusted independently from one another. Such anisotropic barostat is usually a better choice for a solid phase. For a liquid or a gas, the isotropic barostat is usually the best choice.

The simulation takes about 15-20 minutes on 4 cpu cores.

Let us check the evolution of the temperature from the *temperature.dat* file. Apart from an initial spike (may be due to an initial bad configuration, probably harmless here), the temperature follows well the desired annealing procedure.

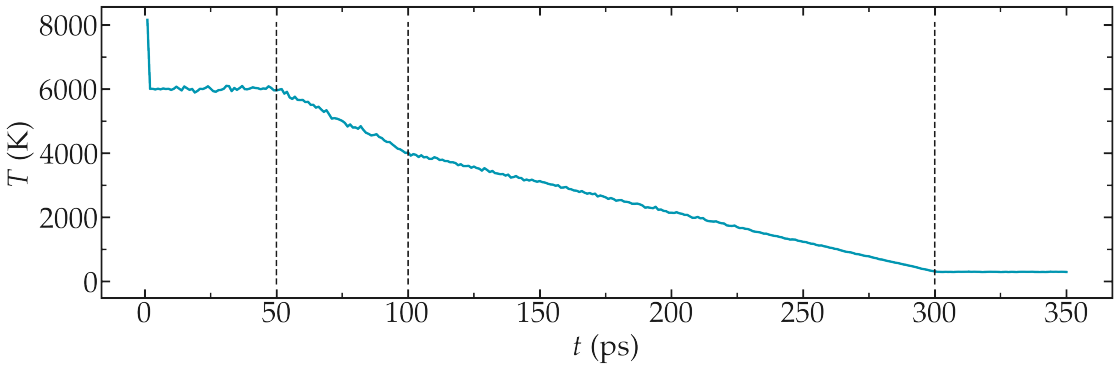


Figure: Temperature of the system during annealing. The vertical dashed lines mark the transition between the different phases of the simulations.

Let us also make sure that the box was indeed deformed isotropically during the first stage of the simulation, and then anisotropically by plotting the evolution of the box dimensions over time.

After running the simulation, the final LAMMPS topology file named *amorphousSiO.data* will be located in *SilicaBlock/*. Alternatively, if you are only interested in the next steps of this tutorial, you can download it by clicking here.

Tip for research project

In the case of a research project, the validity of the generated structure must be tested and compared to reference values, ideally from experiments. For instance, radial distribution functions or Young modulus can both be compared to experimental values. This is beyond the scope of this tutorial.

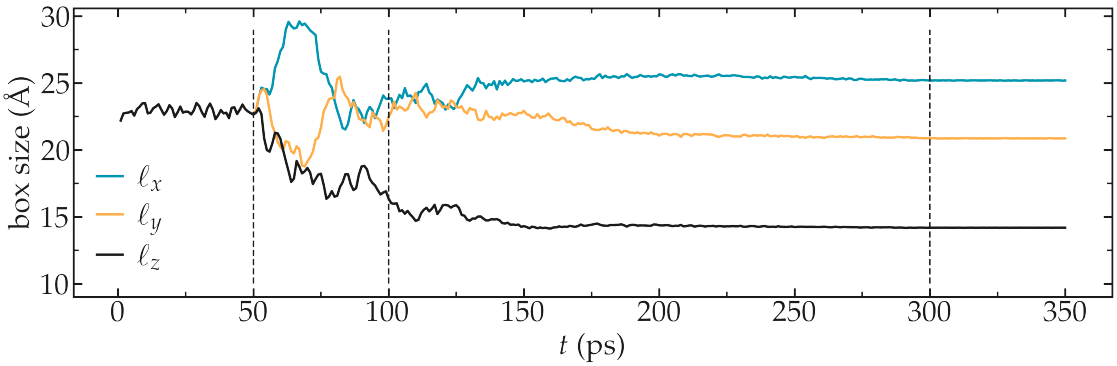


Figure: Box dimensions during annealing. The vertical dashed lines mark the transition between the different phases of the simulations.

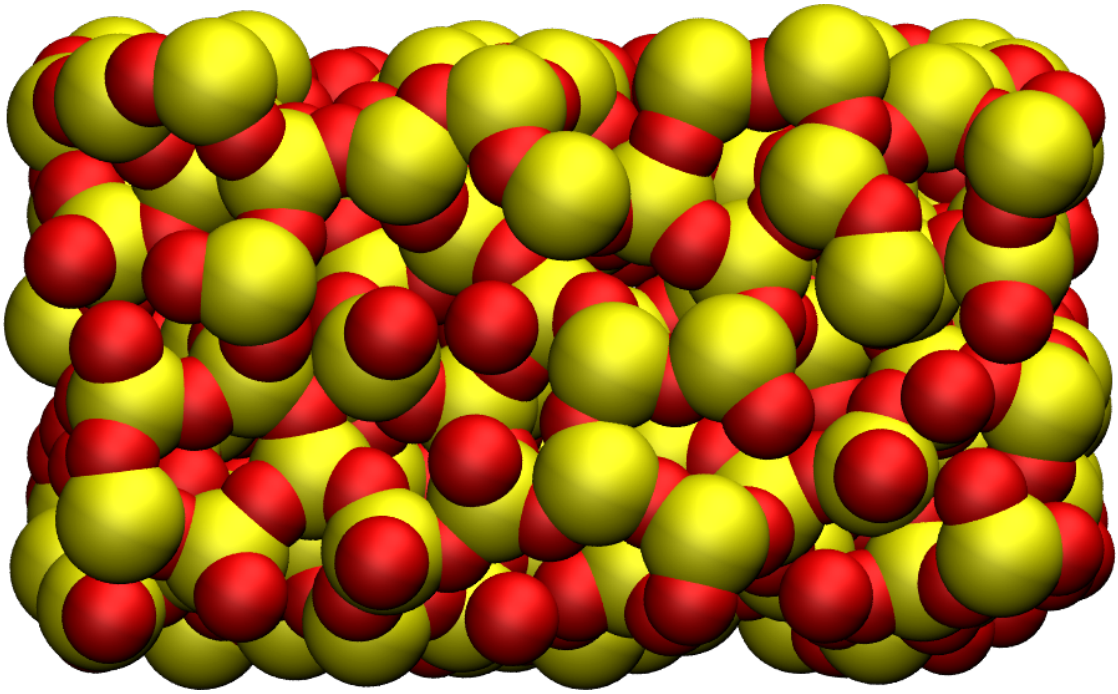


Figure: Snapshot of the final amorphous silica (SiO2) with Si atom in yellow and O atoms in red.

6.2 Cracking the silica

Let us dilate the block of silica to create a crack. Create a new folder called *Cracking/* next to *SilicaBlock/*, and create a new *input.lammps* file starting with familiar lines:

```
units metal
boundary p p p
atom_style full
neighbor 1.0 bin
neigh_modify delay 1

read_data ../SilicaBlock/amorphousSiO.data

pair_style vashishta
pair_coeff * * ../Potential/SiO.1990.vashishta Si O
dump dmp all atom 1000 dump.lammpstrj
```

Then, let us progressively increase the size of the box in the z direction, thus forcing the silica to deform and eventually crack. To do so, let us make a loop using the jump command. At every step of the loop, the box dimension over x will be multiplied by a factor 1.005. Here, we use a NVT thermostat because we want to impose a deformation of the volume.

On using barostat during deformation

Here, box deformations are applied in the x-direction, while the y and z box dimensions are kept constants.

Another possible choice would be to apply a barostat along the y and z direction, allowing the system more freedom to deform. In LAMMPS, this can be done by using :

```
fix npt1 all npt temp 300 300 0.1 y 1 1 1 z 1 1 1
```

instead of:

```
fix nvt1 all nvt temp 300 300 0.1
```

Here, the second option will be used.

Add the following lines to the *input.lammps*:

```
fix nvt1 all nvt temp 300 300 0.1
timestep 0.001
thermo 1000
variable var loop 45
label loop
change_box all x scale 1.005 remap
run 2000
next var
jump input.lammps loop
run 20000
write_data dilatedSiO.data
```

You can use different scale factor (here 1.005) or different number of steps in the loop (here 45) if you want to generate different defects in the silica.

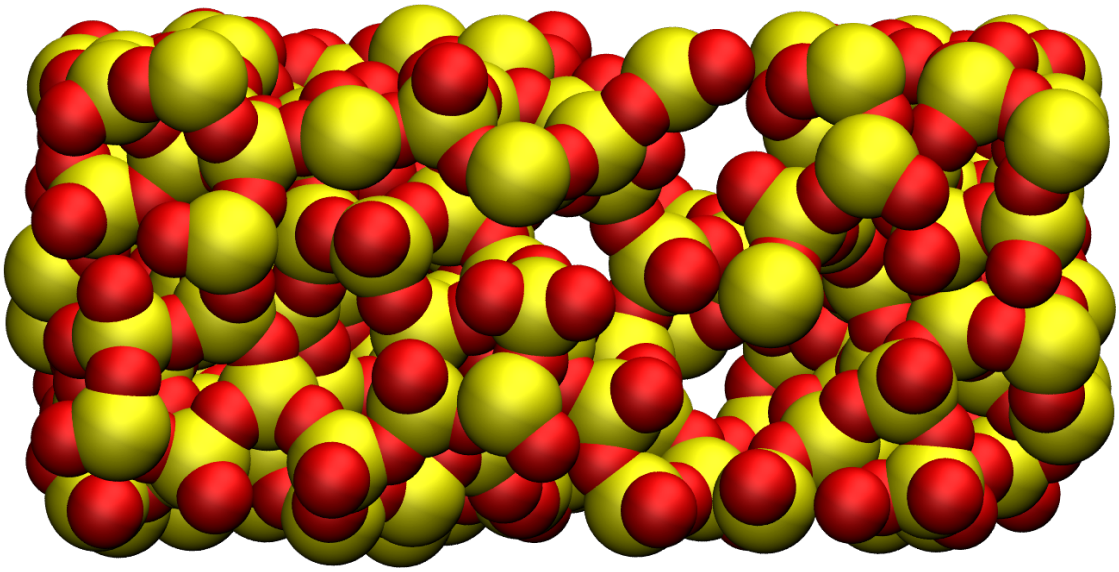


Figure: Block of silica after deformation, with some visible holes.

After the dilatation, a final equilibration step of 20 picoseconds is performed. If you look at the *dump.lammpstrj* file using VMD, you can see the dilatation occurring step-by-step, and the atoms progressively adjusting to the box dimensions.

At first, the deformations are reversible (elastic regime). At some point, bonds start breaking and dislocations appear (plastic regime).

Alternatively, you can download the final state directly by clicking here. The final system with the crack resembles:

Passivated silica

In ambient conditions, Some of the surface SiO2 atoms are chemically passivated by forming covalent bonds with hydrogen (H) atoms. For the sake of simplicity, we are not going to add surface hydrogen atoms here. An example of procedure allowing for properly inserting hydrogen atoms is given as an exercise from Reactive silicon dioxide.

6.3 Adding water

In order to add the water molecules to the silica, we are going to use the Monte Carlo method in the grand canonical ensemble (GCMC). In short, the system is put into contact with a virtual reservoir of given chemical potential μ, and multiple attempts to insert water molecules at random positions are made. Attempts are either accepted or rejected based on energy consideration.

6.3.1 Using hydrid potentials

Create a new folder called *Addingwater/*. Download and save the template file for the water molecule within *Addingwater/*.

Create a new input file called *input.lammps* within *Addingwater/*, and copy the following lines into it:

```
units metal
boundary p p p
atom_style full
neighbor 1.0 bin
neigh_modify delay 1
pair_style hybrid/overlay vashishta lj/cut/tip4p/long 3 4 1 1 0.1546 10
kspace_style pppm/tip4p 1.0e-4
bond_style harmonic
angle_style harmonic
```

There are several differences with the previous input files used in this tutorial. From now on, the system will combine water and silica, and therefore two force fields are combined: Vashishta for SiO, and lj/cut/tip4p/long for TIP4P water model. Combining the two force fields is done using the *hybrid/overlay* pair style.

About hybrid and hybrid/overlay pair style

From the LAMMPS documentation: The hybrid and hybrid/overlay styles enable the use of multiple pair styles in one simulation. With the hybrid style, exactly one pair style is assigned to each pair of atom types. With the hybrid/overlay and hybrid/scaled styles, one or more pair styles can be assigned to each pair of atom types.

The *kspace* solver is used to calculate the long range Coulomb interactions associated with *tip4p/long*. Finally, the style for the bonds and angles of the water molecules are defined, although they are not important since its a rigid water model.

Before going further, we also need to make a few change to our data file. Currently, *dilatedSiO.data* only includes two atom types, but we need four. Copy the previously generated *dilatedSiO.data* file within *Addingwater/*. Currently, *dilatedSiO.data* starts with:

```
576 atoms
2 atom types

-5.512084438507452 26.09766215010596 xlo xhi
-0.12771230207837192 20.71329001367807 ylo yhi
3.211752393088563 17.373825318513106 zlo zhi

Masses

1 28.0855
2 15.9994

Atoms # full

(...)
```

Make the following changes to allow for the addition of water molecules. Modify the file so that it looks like the following (with 4 atom types, 1 bond type, 1 angle type, and four masses):

```
576 atoms
4 atom types
1 bond types
1 angle types

2 extra bond per atom
1 extra angle per atom
2 extra special per atom

0.910777522101565 19.67480018949893 xlo xhi
2.1092682236518137 18.476309487947546 ylo yhi
-4.1701120819606885 24.75568979356097 zlo zhi

Masses

1 28.0855
2 15.9994
3 15.9994
4 1.008

Atoms # full

(...)
```

Doing so, we anticipate that there will be 4 atoms types in the simulations, with O and H of H2O having indexes 3 and 4, respectively. There will also be 1 bond type and 1 angle type. The extra bond, extra angle, and extra special lines are here for memory allocation.

We can continue to fill in the *input.lammps* file, by adding the system definition:

```
read_data dilatedSiO.data
molecule h2omol H2O.mol
lattice sc 3
create_atoms 0 box mol h2omol 45585
lattice none 1

group SiO type 1 2
group H2O type 3 4
```

After reading the data file and defining the h2omol molecule from the txt file, the *create_atoms* command is used to include some water molecules in the system on a simple cubic lattice. Not adding a molecule before starting the GCMC steps usually lead to failure. Note that here, most water molecules are overlapping with the silica. These overlapping water molecules will be deleted before starting the simulation.

Then, add the following settings to *input.lammps*:

```
pair_coeff * * vashishta ../Potential/SiO.1990.vashishta Si O NULL NULL
pair_coeff * * lj/cut/tip4p/long 0 0
# epsilonSi = 0.00403, sigmaSi = 3.69
# epsilonO = 0.0023, sigmaO = 3.091
pair_coeff 1 3 lj/cut/tip4p/long 0.0057 4.42
pair_coeff 2 3 lj/cut/tip4p/long 0.0043 3.12
pair_coeff 3 3 lj/cut/tip4p/long 0.008 3.1589
pair_coeff 4 4 lj/cut/tip4p/long 0.0 0.0
bond_coeff 1 0 0.9572
angle_coeff 1 0 104.52

variable oxygen atom "type==3"
group oxygen dynamic all var oxygen
variable nO equal count(oxygen)
fix myat1 all ave/time 100 10 1000 v_nO file numbermolecule.dat

fix shak H2O shake 1.0e-4 200 0 b 1 a 1 mol h2omol
```

The force field Vashishta applies only to Si (type 1) and O of SiO2 (type 2), and not to the O and H of H2O, thanks to the NULL parameters used for atoms of types 3 and 4.

Pair coefficients for lj/cut/tip4p/long are defined between O atoms, as well as between O(SiO)-O(H2O) and Si(SiO)-O(H2O). Therefore, the fluid-solid interactions will be set by Lennard Jones and Coulomb potentials.

The number of oxygen atoms from water molecules (i.e. the number of molecules) will be printed in the file *numbermolecule.dat*.

The shake algorithm is used to maintain the shape of the water molecules over time. Some of these features have been seen in previous tutorials.

Let us delete the overlapping water molecules, and print the positions of the remaining atoms in a *.lammpstrj* file by adding the following lines to *input.lammps*:

```
delete_atoms overlap 2 H2O SiO mol yes
dump dmp all atom 1000 dump.init.lammpstrj
```

6.3.2 GCMC simulation

Just before starting the GCMC simulation, let us make a first equilibration step by adding the following lines to *input.lammps*:

```
compute_modify thermo_temp dynamic yes
compute ctH2O H2O temp
compute_modify ctH2O dynamic yes
fix mynvt1 H2O nvt temp 300 300 0.1
fix_modify mynvt1 temp ctH2O
compute ctSiO SiO temp
fix mynvt2 SiO nvt temp 300 300 0.1
fix_modify mynvt2 temp ctSiO
timestep 0.001
thermo 1000
run 5000
```

On thermostating groups instead of the entire system

Two different thermostats are used for SiO and for H2O, respectively. Using separate thermostats is usually better when the system contains two separate species, such as a solid and a liquid. It is particularly important to use two thermostats here because the number of water molecules will fluctuate with time.

The *compute_modify* with *dynamic yes* for water is used to specify that the number of molecules is not constant.

Finally, let us use the *fix gcmc* and perform the grand canonical Monte Carlo steps. Add the following lines into *input.lammps*:

```
variable tfac equal 5.0/3.0
variable xlo equal xlo+0.1
variable xhi equal xhi-0.1
variable ylo equal ylo+0.1
variable yhi equal yhi-0.1
variable zlo equal zlo+0.1
variable zhi equal zhi-0.1
region system block ${xlo} ${xhi} ${ylo} ${yhi} ${zlo} ${zhi}
fix fgcmc H2O gcmc 100 100 0 0 65899 300 -0.5 0.1 &
    mol h2omol tfac_insert ${tfac} group H2O shake shak &
    full_energy pressure 10000 region system
run 45000
write_data SiOwithwater.data
write_dump all atom dump.lammpstrj
```

Dirty fix

The region *system* was created to avoid the error Fix gcmc region extends outside simulation box which seems to occur with the 2Aug2023 LAMMPS version.

The *tfac_insert* option ensures that the correct estimate is made for the temperature of the inserted water molecules by taking into account the internal degrees of freedom. Running this simulation, you should see the number of molecules increasing progressively. When using the pressure argument, LAMMPS ignores the value of the chemical potential [here $\mu = -0.5\,\text{eV}$ which corresponds roughly to ambient conditions (i.e. RH $\approx 50\,\%$).] The large pressure value of 10000 bars was chosen to ensure that some successful insertions of molecules would occur during the extremely short duration of this simulation.

When you run the simulation, make sure that some water molecules remain in the system after the *delete_atoms* command. You can control that either using the log file, or using the *numbermolecule.dat* data file.

You can see, by looking at the log file, that 280 molecules were added by the *create_atoms* command (the exact number you get may differ):

```
Created 840 atoms
```

You can also see that 258 molecules where immediately deleted, leaving 24 water molecules (the exact number you get may differ):

```
Deleted 774 atoms, new total = 642
Deleted 516 bonds, new total = 44
Deleted 258 angles, new total = 22
```

After just a few GCMC steps, the number of molecules starts increasing with time. Once the crack is fully filled with water molecules, the number of molecules reaches a plateau.

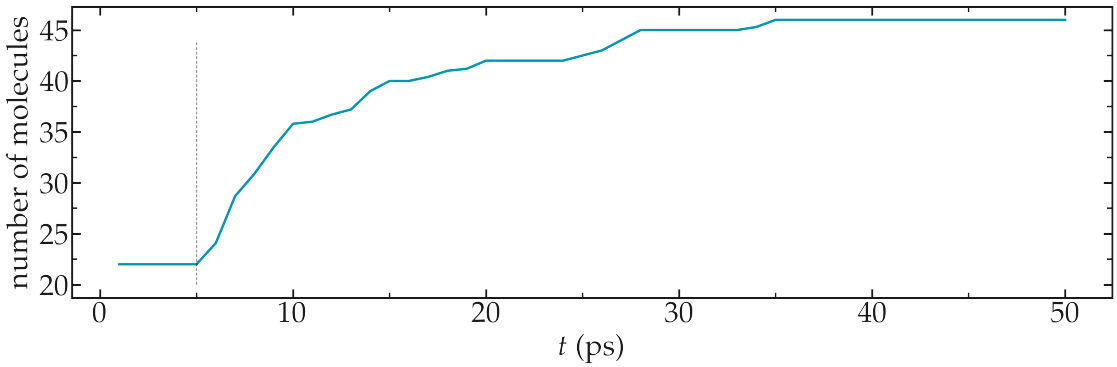


Figure: Number of molecules as a function of time. The dashed vertical line marks the beginning of the GCMC step.

Note that the final number of molecules depends on the imposed pressure, temperature, and on the interaction between water and silica (i.e. its hydrophilicity).

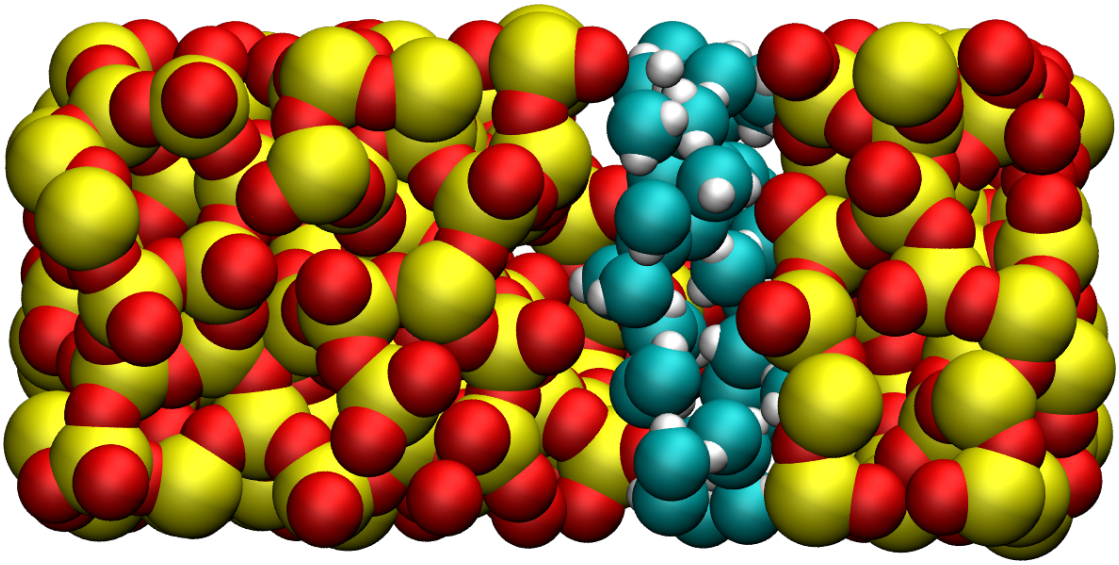


Figure: Snapshot of the silica system after the adsorption of the water molecules, with the oxygen of the water molecules represented in cyan.

Note that GCMC simulations of such dense phases are usually slow to converge due to the very low probability of successfully inserting a molecule. Here, the short simulation duration was made possible by the use of a large pressure.

Vizualising varying number of molecules

By default, VMD fails to properly render systems with varying number of atoms.

You can access all the input scripts and data files that are used in these tutorials from the inputs folder on Github. This repository also contains the full solutions to the exercises.

6.4 Going further with exercises

Each exercise comes with a proposed solution, see Solutions to the exercises.

6.4.1 Mixture adsorption

Adapt the existing script and insert both CO_2 molecules and water molecules within the silica crack using GCMC. Download the CO2 template. The parameters for the CO_2 molecule are the following:

```
pair_coeff 5 5 lj/cut/tip4p/long 0.0179 2.625854
pair_coeff 6 6 lj/cut/tip4p/long 0.0106 2.8114421
bond_coeff 2 46.121 1.17
angle_coeff 2 2.0918 180
```

Where is is assumed that the atom of type 5 is oxygen of mass 15.9994, and atom of type 6 is carbon of mass 12.011.

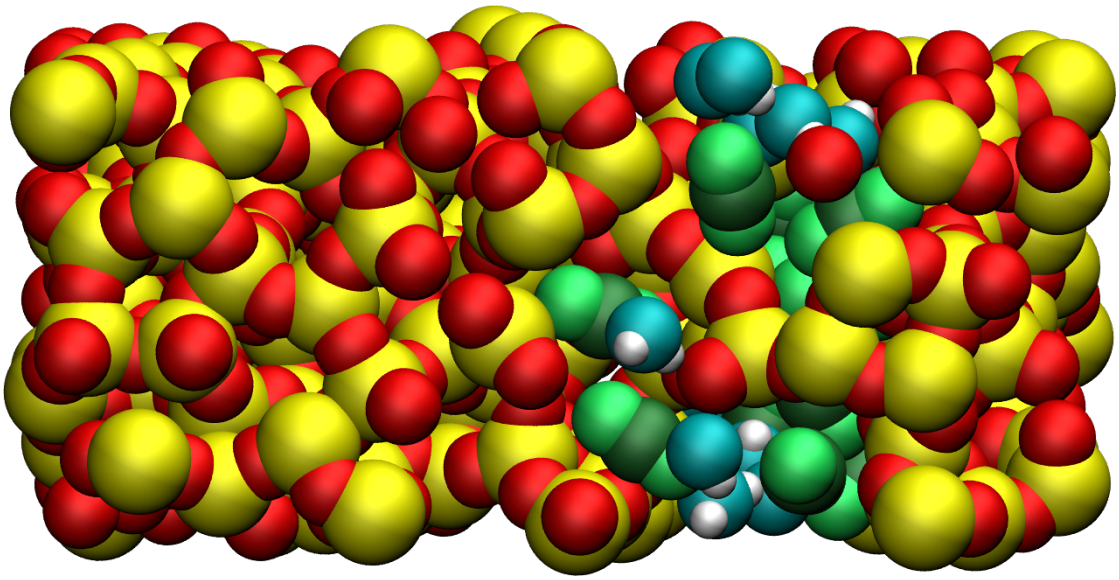
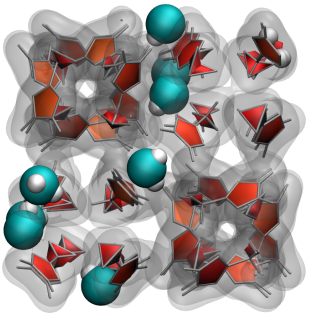


Figure: Cracked silica with adsorbed water and CO_2 molecules (in green).

6.4.2 Adsorb water in ZIF-8 nanopores



Use the same protocole as the one implemented in this tutorial to add water molecules to a Zif-8 nanoporous material. A snapshot of the system with a few water molecules is shown on the right.

Download the initial Zif-8 structure, the parameters file, and this new water template. The ZIF-8 structure is made of 7 atom types (C1, C2, C3, H2, H3, N, Zn), connected by bonds, angles, dihedrals, and impropers. It uses the same *pair_style* as water, so there is no need to use *hybrid pair_style*. Your *input* file should start like that:

```
units real
atom_style full
boundary p p p
bond_style harmonic
angle_style harmonic
dihedral_style charmm
improper_style harmonic

pair_style lj/cut/tip4p/long 1 2 1 1 0.105 14.0
kspace_style pppm/tip4p 1.0e-5

special_bonds lj 0.0 0.0 0.5 coul 0.0 0.0 0.833
```

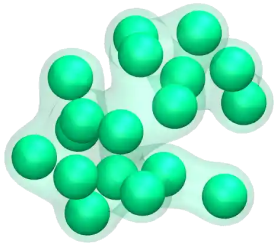
Note that, here, water occupies the atom types 1 and 2, instead of 3 and 4 in the case of SiO2.

7

Free energy calculation

Sampling a free energy barrier

The objective of this tutorial is to measure a free energy profile of particles across a barrier potential using two methods; free sampling and umbrella sampling.



For the sake of simplicity and in order to reduce the computation time, the barrier potential will be imposed artificially to the atoms. The procedure is valid for more complex systems, and can be adapted to many other situations, for instance for measuring adsorption barrier near a wall, or for calculating translocation barrier through a membrane.

If you are completely new to LAMMPS, I recommend you to follow this tutorial on a simple Lennard Jones fluid first.

What is free energy

The *free energy* refers to the potential energy of a system that is available to perform work. In molecular simulations, it is common to calculate free energy differences between different states or conformations of a molecular system. This can be useful in understanding the thermodynamics of a system, predicting reaction pathways, and determining the stability of different molecular configurations.

7.1 Method 1: Free sampling

The most direct way to calculate a free energy profile is to extract the partition function from a classic (unbiased) molecular dynamics simulation, and then to estimate the Gibbs free energy using

$$\Delta G = -RT \ln(p/p_0),$$

where ΔG is the free energy difference, R the gas constant, T the temperature, p the pressure, and p_0 the reference pressure. As an illustration, let us apply this method to an extremely simple configuration that consists in a few particles diffusing in a box in presence of a position-dependent repealing force that makes the centre of the box a relatively unfavourable area to explore.

7.1.1 Basic LAMMPS parameters

Create a folder named *FreeSampling/*, and create an input script named *input.lammps* in it. Copy the following lines into it:

```
variable sigma equal 3.405 # Angstrom
variable epsilon equal 0.238 # Kcal/mol
variable U0 equal 1.5*${epsilon} # Kcal/mol
variable dlt equal 1.0 # Angstrom
variable x0 equal 10.0  # Angstrom

units real
atom_style atomic
pair_style lj/cut 3.822
pair_modify shift yes
boundary p p p
```

Here, we start by defining variables for the Lennard-Jones interaction σ and ϵ and for the repulsive potential $U(x)$: U_0, δ, and x_0, see the analytical expression below.

The value of 3.822 for the cut off was chosen to create a WCA, purely repulsive, potential. It was calculated as $2^{1/6} \times 3.405$ where $3.405 = \sigma$.

The system of unit '*real*, for which energy is in kcal/mol, distance in Ångstrom, time in femtosecond, has been chosen for practical reason: the WHAM algorithm used in the second part of the tutorial automatically assumes the energy to be in kcal/mol. Atoms will interact through a Lennard-Jones potential with a cut-off equal to $\sigma \times 2^{1/6}$ (i.e. a WCA repulsive potential). The potential is shifted to be equal to 0 at the cut-off using the *pair_modify*.

7.1.2 System creation and settings

Let us define the simulation block and randomly add atoms by adding the following lines to *input.lammps*:

```
region myreg block -25 25 -5 5 -25 25
create_box 1 myreg
create_atoms 1 random 60 341341 myreg overlap 1.0 maxtry 50

mass * 39.95
pair_coeff * * ${epsilon} ${sigma}
neigh_modify every 1 delay 4 check yes
```

Here, I am using the argon's values of the Lennard-Jones parameters σ and ϵ, as well as the mass $m = 39.95$ grams/mole.

In the previous subsection, the variables U_0, δ, and x_0 were defined. They are used to create the repulsive potential restricting the atoms to explore the center of the box:

$$U(x) = U_0 \left[\arctan\left(\frac{x + x_0}{\delta}\right) - \arctan\left(\frac{x - x_0}{\delta}\right) \right].$$

From the derivative of the potential with respect to x, we obtain the expression for the force that will be imposed to the atoms:

$$F(x) = \frac{U_0}{\delta} \left[\frac{1}{(x - x_0)^2/\delta^2 + 1} - \frac{1}{(x + x_0)^2/\delta^2 + 1} \right].$$

The potential and force along the x axis resemble:

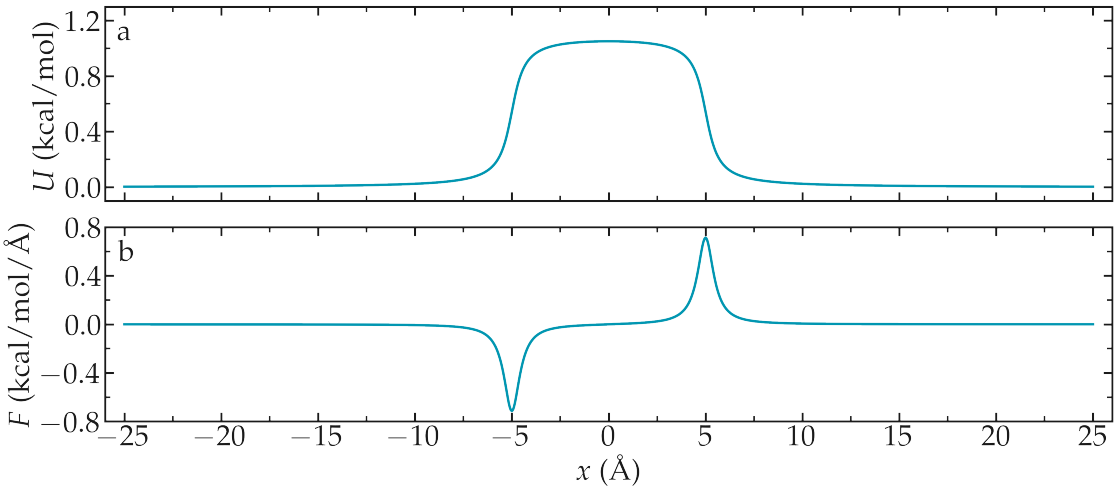


Figure: a) Potential $U(x)$. b) force $F(x)$ (bottom) imposed to the atoms.

Let us apply energy minimization to the system, and then impose the force $F(x)$ to all of the atoms in the simulation using the *addforce* command. Add the following lines to *input.lammps*:

```
minimize 1e-4 1e-6 100 1000
reset_timestep 0

variable U atom ${U0}*atan((x+${x0})/${dlt}) &
    -${U0}*atan((x-${x0})/${dlt})
variable F atom ${U0}/((x-${x0})^2/${dlt}^2+1)/${dlt} &
    -${U0}/((x+${x0})^2/${dlt}^2+1)/${dlt}
fix myadf all addforce v_F 0.0 0.0 energy v_U
```

Finally, let us combine the *fix nve* with a *Langevin* thermostat and run a molecular dynamics simulation. With these two commands, the MD simulation is effectively in the NVT ensemble:

constant number of atoms N, constant volume V, and constant temperature T. Let us perform an equilibration of 500000 steps in total, using a timestep of 2 fs (i.e. a total duration of 1 ns).

To make sure that 1 ns is long enough, let us record the evolution of the number of atoms in the central (energetically unfavorable) region called *mymes*:

```
fix mynve all nve
fix mylgv all langevin 119.8 119.8 50 1530917

region mymes block -${x0} ${x0} INF INF INF INF
variable n_center equal count(all,mymes)
fix myat all ave/time 10 50 500 v_n_center file density_evolution.dat

timestep 2.0
thermo 10000
run 500000
```

7.1.3 Run and data acquisition

Finally, let us record the density profile of the atoms along the x axis using the *ave/chunk* command. A total of 10 density profiles will be printed. The step count is reset to 0 to synchronize with the output times of *fix density/number*, and the *fix myat* is canceled (it has to be canceled before a reset time):

```
unfix myat
reset_timestep 0

compute cc1 all chunk/atom bin/1d x 0.0 1.0
fix myac all ave/chunk 10 400000 4000000 &
    cc1 density/number file density_profile_8ns.dat
dump mydmp all atom 200000 dump.lammpstrj

thermo 100000
run 4000000
```

This simulation with a total duration of 9 ns needs a few minutes to complete. Feel free to increase the duration of the last run for smoother results.

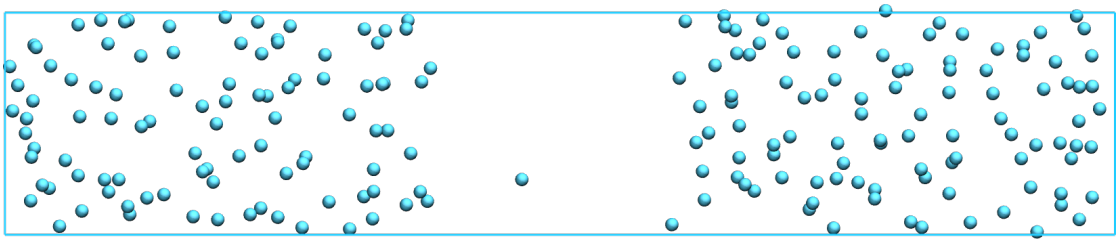


Figure: Notice that the density of atoms is lower in the central part of the box, due to the additional force $F(x)$.

7.1.4 Data analysis

First, let us make sure that the initial equilibration of 1 ns is long enough by looking at the *density_evolution.dat* file.

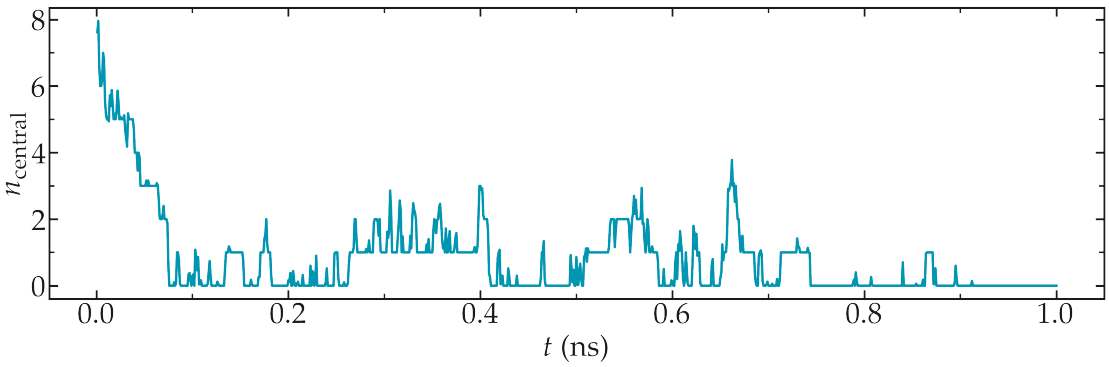


Figure: Evolution of the number of atoms in the central region during equilibration.

Here, we can clearly see that the number of atoms in the central region, n_{central}, evolves near its equilibrium value (which is close to 0) after about 0.1 ns.

One can also have a look at the density profile, which shows that the density in the center of the box is about two orders of magnitude lower than inside the reservoir.

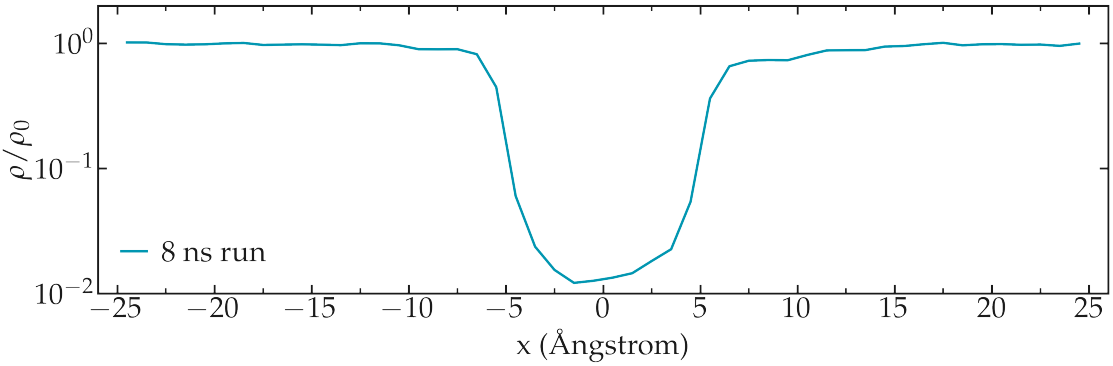


Figure: Averaged density profiles for the 8 ns run. The value for the reference density $\rho_{\mathrm{bulk}} = 0.0033$ was estimated from the raw density profiles.

Then, let us plot $-RT\ln(\rho/\rho_{\mathrm{bulk}})$ and compare it with the imposed (reference) potential U.

The agreement with the expected energy profile is reasonable, despite some noise in the central part.

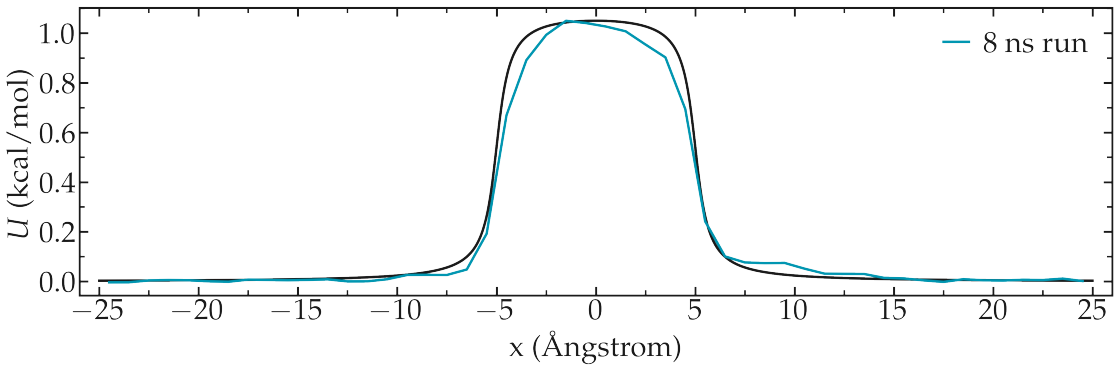
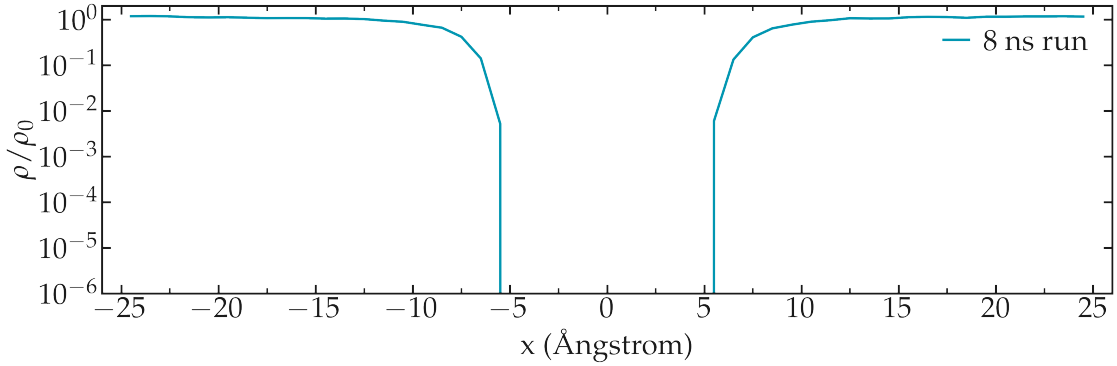


Figure: Calculated potential $-RT\ln(\rho/\rho_{\text{bulk}})$ compared to imposed potential. The calculated potential is in blue.

7.1.5 The limits of free sampling

If we increase the value of U_0, the average number of atoms in the central region will decrease, making it difficult to obtain a good resolution for the free energy profile. For instance, using $U_0 = 10\epsilon$, not a single atom crosses the central part of the simulation, despite the 8 ns of simulation.



In that case, it is better to use the umbrella sampling method to extract free energy profiles, see the next section.

7.2 Method 2: Umbrella sampling

Umbrella sampling is a biased molecular dynamics method, i.e. a method in which additional forces are added to the atoms in order to make the unfavourable states more likely to occur.

Several simulations (or windows) will be performed with different parameters for the imposed biasing.

Here, let us force one of the atom to explore the central region of the box. To do so, let us add a

potential V to one of the particle, and force it to move along the axis x. The chosen path is called the axis of reaction. The results will be analyzed using the weighted histogram analysis method (WHAM), which allows to remove the effect of the bias and eventually deduce the unbiased free energy profile.

7.2.1 LAMMPS input script

Create a new folder called *BiasedSampling/*, create a new input file named *input.lammps* in it, and copy the following lines:

```
variable sigma equal 3.405 # Angstrom
variable epsilon equal 0.238 # Kcal/mol
variable U0 equal 10*${epsilon} # Kcal/mol
variable dlt equal 0.5 # Angstrom
variable x0 equal 5.0  # Angstrom
variable k equal 1.5 # Kcal/mol/Angstrom^2

units real
atom_style atomic
pair_style lj/cut 3.822 # 2^(1/6) * 3.405 WCA potential
pair_modify shift yes
boundary p p p

region myreg block -25 25 -5 5 -25 25
create_box 2 myreg
create_atoms 2 single 0 0 0
create_atoms 1 random 5 341341 myreg overlap 1.0 maxtry 50

mass * 39.948
pair_coeff * * ${epsilon} ${sigma}
neigh_modify every 1 delay 4 check yes
group topull type 2

variable U atom ${U0}*atan((x+${x0})/${dlt}) &
    -${U0}*atan((x-${x0})/${dlt})
variable F atom ${U0}/((x-${x0})^2/${dlt}^2+1)/${dlt} &
    -${U0}/((x+${x0})^2/${dlt}^2+1)/${dlt}
fix pot all addforce v_F 0.0 0.0 energy v_U

fix mynve all nve
fix mylgv all langevin 119.8 119.8 50 1530917
timestep 2.0
thermo 100000
run 500000
reset_timestep 0

dump mydmp all atom 1000000 dump.lammpstrj
```

So far, this code resembles the one of Method 1, except for the additional particle of type 2. This particle is identical to the particles of type 1 (same mass and Lennard-Jones parameters), but will be exposed to the biasing potential.

Let us create a loop with 50 steps, and move progressively the centre of the bias potential by increment of 0.1 nm. Add the following lines into *input.lammps*:

```
variable a loop 50
label loop
variable xdes equal ${a}-25
variable xave equal xcm(topull,x)
fix mytth topull spring tether ${k} ${xdes} 0 0 0
run 200000
fix myat1 all ave/time 10 10 100 v_xave v_xdes &
    file data-k1.5/position.${a}.dat
run 1000000
unfix myat1
next a
jump SELF loop
```

A folder named *data-k1.5/* needs to be created within *BiasedSampling/*.

The spring command serves to impose the additional harmonic potential with spring constant k. Note that the value of k should be chosen with care, if k is too small, the particle wont follow the biasing potential center, if k is too large, there will be no overlapping between the different windows. See the side note named *on the choice of k* below.

The centre of the harmonic potential x_{des} successively takes values from -25 to 25. For each value of x_{des}, an equilibration step of 0.4 ns is performed, followed by a step of 2 ns during which the position along x of the particle is saved in data files (one data file per value of x_{des}). You can always increase the duration of the runs for better samplings.

7.2.2 WHAM algorithm

In order to generate the free energy profile from the density distribution, we are going to use the WHAM algorithm.

You can download it from Alan Grossfield website, and compile it using:

```
cd wham
make clean
make
```

The compilation creates an executable called *wham* that you can copy in the *BiasedSampling/* folder. Alternatively, use the version 2.0.11 I have downloaded, or try your luck with the version i did precompile; precompiled wham.

In order to apply the WHAM algorithm to our simulation, we first need to create a metadata file. This file simply contains

- the paths to all the data files,
- the value of x_{des},

- and the values of k.

To generate the *metadata.txt* file, you can run this Python script from the *BiasedSampling/* folder:

```
import os

k=1.5
folder='data-k1.5/'

f = open("metadata.dat", "w")
for n in range(-50,50):
    datafile=folder+'position.'+str(n)+'.dat'
    if os.path.exists(datafile):
        # read the imposed position is the expected one
        with open(datafile) as g:
            _ = g.readline()
            _ = g.readline()
            firstline = g.readline()
        imposed_position = firstline.split(' ')[-1][:-1]
        # write one file per file
        f.write(datafile+' '+str(imposed_position)+' '+str(k)+'\n')
f.close()
```

Here k is in kcal/mol. The generated file named *metadata.dat* looks like that:

```
./data-k1.5/position.1.dat -24 1.5
./data-k1.5/position.2.dat -23 1.5
./data-k1.5/position.3.dat -22 1.5
(...)
./data-k1.5/position.48.dat 23 1.5
./data-k1.5/position.49.dat 24 1.5
./data-k1.5/position.50.dat 25 1.5
```

Alternatively, you can download this metadata.dat file. Then, simply run the following command in the terminal:

```
./wham -25 25 50 1e-8 119.8 0 metadata.dat PMF.dat
```

where -25 and 25 are the boundaries, 50 the number of bins, 1e-8 the tolerance, and 119.8 the temperature. A file named PMF.dat has been created, and contains the free energy profile in Kcal/mol.

Again, one can compare the result of the PMF with the imposed potential U, which shows that the agreement is excellent.

We can see that the agreement is quite good despite the very short calculation time and the very high value for the energy barrier. Obtaining the same results with Free Sampling would require to perform extremely long and costly simulations.

You can access all the input scripts and data files that are used in these tutorials from the inputs folder on Github. This repository also contains the full solutions to the exercises.

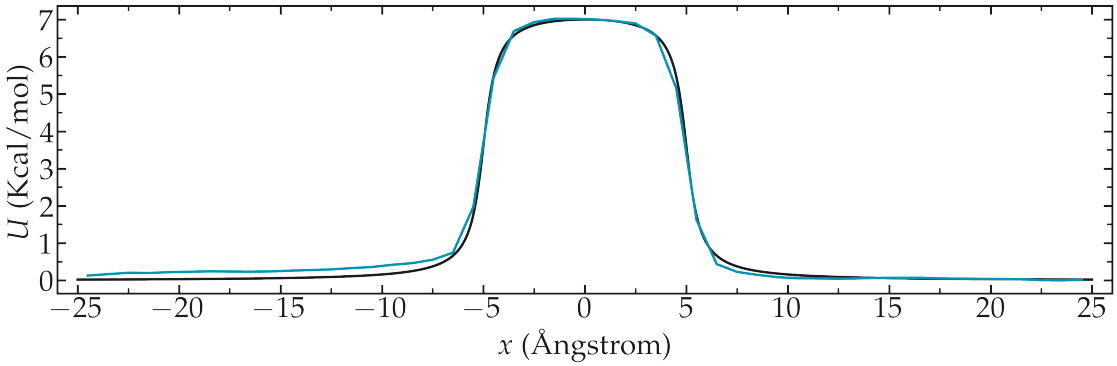


Figure: Calculated potential using umbrella sampling compared to the imposed potential. The calculated potential is in blue.

7.2.3 Side note: on the choice of k

As already stated, one difficult part of umbrella sampling is to choose the value of k. Ideally, you want the biasing potential to be strong enough to force the chosen atom to move along the axis, and you also want the fluctuations of the atom position to be large enough to have some overlap in the density probability of two neighbor positions. Here, 3 different values of k are being tested.

If k is too small, the biasing potential is too weak to force the particle to explores the region of interest, making it impossible to reconstruct the PMF.

If k is too large, the biasing potential is too large compared to the potential one want to probe, which reduces the sensitivity of the method.

7.3 Going further with exercises

Each exercise comes with a proposed solution, see Solutions to the exercises.

7.3.1 The binary fluid that wont mix

1 - Create the system

Create a molecular simulation with two species of respective types 1 and 2. Apply different potentials $U1$ and $U2$ on particles of types 1 and 2, respectively, so that particles of type 1 are excluded from the center of the box, while at the same time particles of type 2 are excluded from the rest of the box.

2 - Measure the PMFs

Using the same protocole as the one used in the tutorial (i.e. umbrella sampling with the wham algorithm), extract the PMF for each particle type.

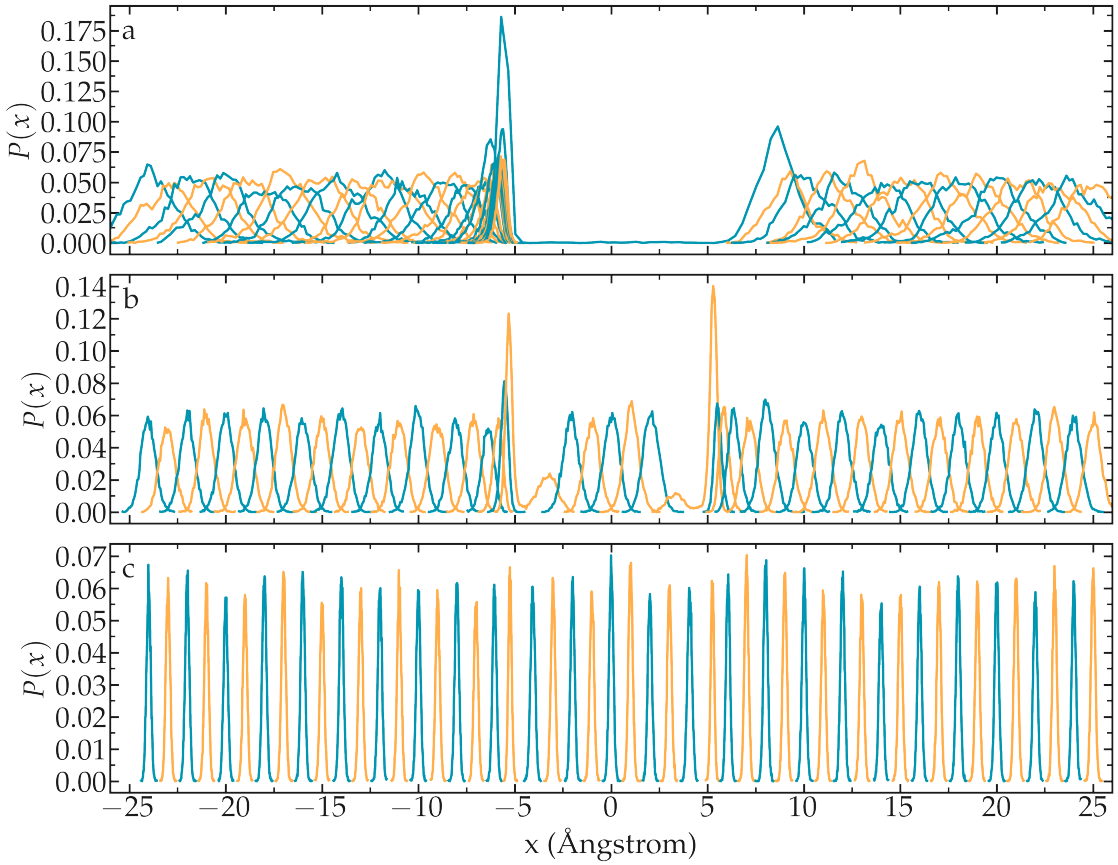


Figure: Density probability for each run with $k = 0.15\,\text{Kcal/mol/Å}^2$ (a), $k = 1.5\,\text{Kcal/mol/Å}^2$ (b), and $k = 15\,\text{Kcal/mol/Å}^2$ (c).

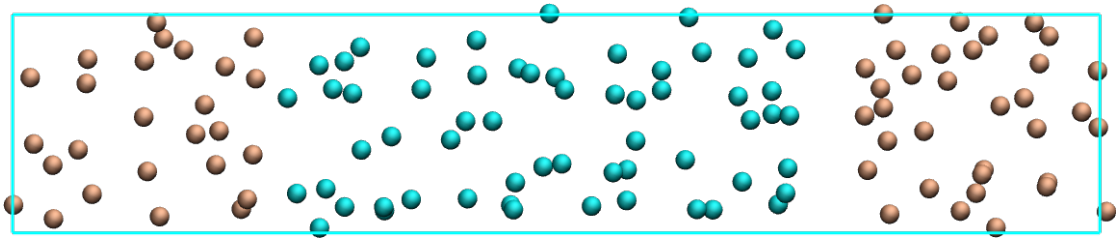


Figure: Particles of type 1 and 2 separated by two different potentials.

7.3.2 Particles under convection

Use a similar simulation as the one from the tutorial, with a repulsive potential in the center of the box. Add an additional forcing to the particles and force them to flow in the x direction.

Re-measure the potential in presence of the flow, and observe the difference with the reference case in absence of flow.

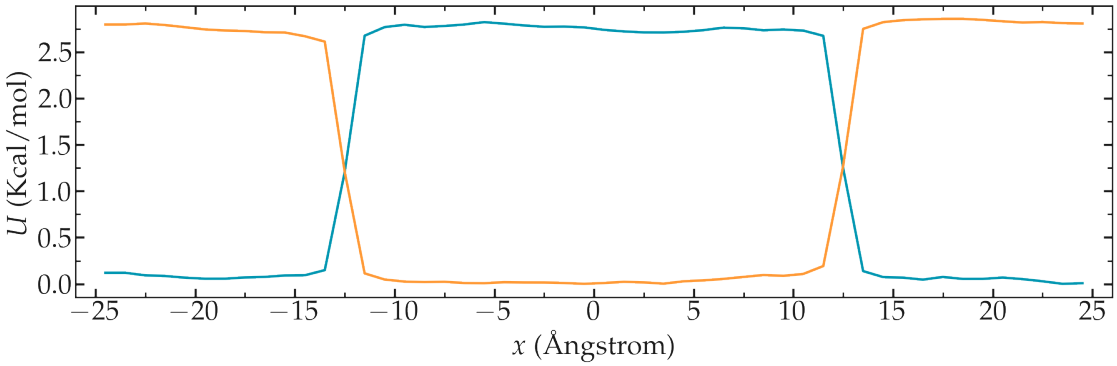


Figure: PMFs calculated for both atom types.

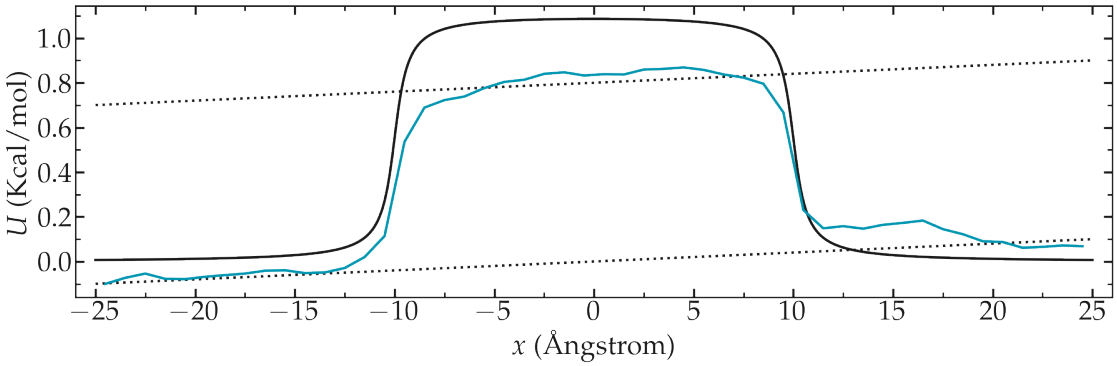


Figure: PMF calculated in the presence of a net forcing inducing the convection of the particles from left to right.

7.3.3 Surface adsorption of a molecule

Apply umbrella sampling to calculate the free energy profile of ethanol in the direction normal to a crystal solid surface (here made of sodium chloride). Find the topology files and parameter file.

Use the following lines for starting the *input.lammps*:

```
units real # style of units (A, fs, Kcal/mol)
atom_style full # molecular + charge
bond_style harmonic
angle_style harmonic
dihedral_style harmonic
boundary p p p # periodic boundary conditions
pair_style lj/cut/coul/long 10 # cut-off 1 nm
kspace_style pppm 1.0e-4
pair_modify mix arithmetic tail yes
```

The PMF normal to a solid wall serves to indicate the free energy of adsorption, which can be calculated from the difference between the PMF far from the surface, and the PMF at the wall.

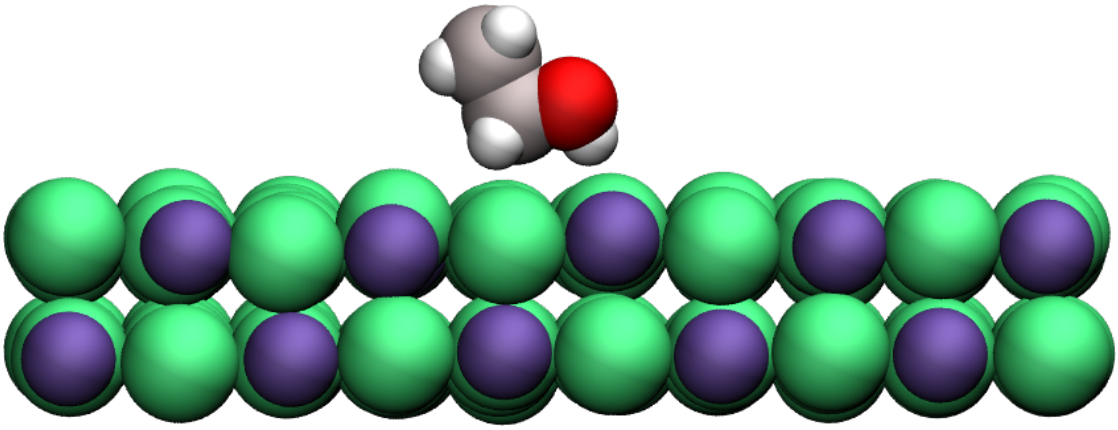


Figure: A single ethanol molecule next to a crystal NaCl(100) wall.

The PMF shows a mimina near the solid surface, which indicates a good affinity between the wall and the molecule.

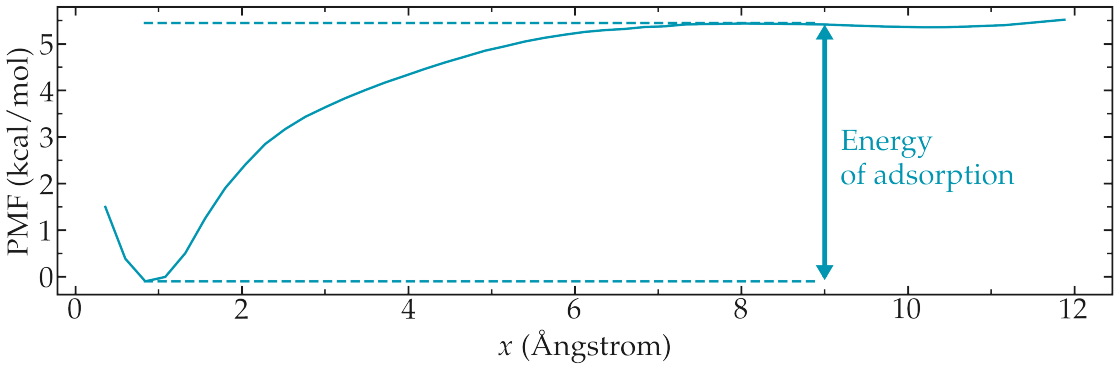


Figure: PMF for a single ethanol molecule next to a NaCl solid surface. The position of the wall is in $x = 0$. The arrow highlight the difference between the energy of the molecule when adsorbed to the solid surface, and the energy far from the surface. This difference corresponds to the free energy of adsorption.

Alternatively to using ethanol, feel free to download the molecule of your choice, for instance from the Automated Topology Builder (ATB). Make your life simpler by choosing a small molecule like CO2.

Solutions to the exercises

Lennard Jones fluid

Fix a broken input

You can download the working input I wrote.

The trick to make the simulation starts without error is to reduce the initial *timestep* value as well as the imposed *temperature*.

```
fix mylgv all langevin 0.001 0.001 0.001 1530917
timestep 0.0001
```

Note that in order to make sure that the temperature of the particles quickly reaches a reasonable value, the *damping* parameter of the *fix Langevin* was also reduced.

With these commands, you should see that after the first *run* finishes, the energy of the system has reduced. Thus, a second *run* with the original *timestep* and *Langevin* parameters can start without issue.

In some cases, more than two consecutive *runs* can be an appropriate solution:

```
fix mylgv all langevin 0.0001 0.0001 0.001 1530917
timestep 0.0001
run 10000

timestep 0.001
run 10000

timestep 0.01
run 10000
```

Use trial and error to determine the best approach for a given system.

Create a demixed dense phase

You can download the input I wrote. Note that I use a large number of particles: 8000 for each type.

The key to create a demixing phase is to play with the Lennard-Jones parameters:

```
pair_coeff 1 1 5.0 1.0
pair_coeff 2 2 5.0 1.0
pair_coeff 1 2 0.05 1.0
```

First, notice that both particle types have the same σ value of 1.0 so that both particles have the same diameter. Second, note the large energy parameter $\epsilon = 5.0$ for self interaction (i.e.) interaction between particles of same type, and the low energy parameter $\epsilon = 0.05$ for interaction between particles of different types.

Finally, for adjusting the box volume and create a liquid looking phase, the pressure was imposed by replacing *fix nve* by *fix nph*:

```
fix mynph all nph iso 1.0 1.0 1.0
```

With *fix nph* and a pressure of 1, LAMMPS adjusts the box dimensions until the pressure is close to 1. Here, reaching a pressure of 1 requires reducing the initial box dimensions.

From atoms to molecules

You can download the input I wrote to make dumbell molecules.

The first important change is to choose an *atom_style* that allows for the atoms to be connected by bonds. It is also necessary to specify the *bond_style*, i.e. the type of potential (here harmonic) that will keep the atoms together:

```
atom_style molecular
bond_style harmonic
```

When creating the box, it is necessary to make memory space for the bond:

```
create_box 2 simulation_box bond/types 1 extra/bond/per/atom 1
```

Then, import the *molecule template*, and use the template when creating the atoms:

```
molecule dumbell dumbell.mol
create_atoms 1 random 500 341341 simulation_box
create_atoms 0 random 5 678865 simulation_box mol dumbell 8754
```

You can download the molecule template by clicking here. Finally, some parameters for the bond, namely its rigidity (5) and equilibrium length (2.5) need to be specified:

```
bond_coeff 1 5 2.5
```

You can download the input and molecule template I wrote to make the short polymer. Note that some additional angular potentials are used to give its rigidity to the polymer.

Pulling on a carbon nanotube

Plot the strain-stress curves

You can download the input for the breakable CNT and input for the unbreakable CNT I wrote.

The stress is calculated as the total force induced on the CNT by the pulling divided by the surface area of the CNT.

On the side note, the surface area of a CNT is not a well defined quantity. Here, I choose to define the area as the perimeter of the CNT multiplied by the effective width of the carbon atoms.

Be careful with units, as the force is either in kcal/mol/Å when the unit is *real*, i.e. for the unbreakable CNT, or in eV/Å when the unit is *metal*, i.e. for the breakable CNT.

Solve the flying ice cube artifact

The issue occurs because the atoms have a large momentum in the x direction, as can be seen by looking at the net velocity of the atoms in the *cnt_molecular.data* file.

```
Velocities

24 0.007983439029626362 -6.613056392124822e-05 7.867644943646289e-05
1 0.007906200203484036 3.252025147011299e-05 -4.4209216231039336e-05
25 0.007861090484107148 9.95045322688365e-06 -0.00014277147407215768
(...)
```

The Berendsen thermostat is trying to adjust the temperature of the system by rescaling the velocity of the atoms, but fails due to the large momentum of the system that makes it look like the system is warm, since in MD temperature is measured from the kinetic energy.

This leads to the system appearing frozen.

The solution is to cancel the net momentum of the atoms, for instance by using *fix momentum*, re-setting the velocity with the *velocity create* command, or use a different thermostat.

Insert gas in the carbon nanotube

You can download the input I wrote.

The key is to modify the *.data* file to make space for the second atom type 2.

```
670 impropers
2 atom types
1 bond types

(...)

Masses

1 12.010700 # CA
2 39.948 # Ar
```

The *parm.lammps* must contain the second pair coeff:

```
pair_coeff 1 1 0.066047 3.4
pair_coeff 2 2 0.232 3.3952
bond_coeff 1 469 1.4
```

Combine the *region* and *create_atoms* commands to create the atoms of type 2 within the CNT:

```
region inside_CNT cylinder z 0 0 2.5 ${zmin} ${zmax}
create_atoms 2 random 40 323485 inside_CNT overlap 1.8 maxtry 50
```

It is good practice to thermalize the CNT separately from the gas to avoid having large temperature difference between the two type of atoms.

```
compute tcar carbon_atoms temp
fix myber1 all temp/berendsen ${T} ${T} 100
fix_modify myber1 temp tcar
compute tgas gas temp
fix myber2 all temp/berendsen ${T} ${T} 100
fix_modify myber2 temp tgas
```

Here I also choose to keep the CNT near its original position,

```
fix myspr carbon_atoms spring/self 5
```

Make a membrane of CNTs

You can download the input I wrote.

The CNT can be replicated using the *replicate* command. It is recommended to adjust the box size before replicating, as done here using the *change_box* command.

To allow for the deformation of the box along the *xy* plane, the box has to be changed to triclinic first:

```
change_box all triclinic
```

Deformation can be imposed to the system using:

```
fix muyef all deform 1 xy erate 5e-5
```

Polymer in water

Extract radial distribution function

You can download the input file I wrote.

I use the *compute rdf* command of LAMMPS to extract the RDF between atoms of type 8 (oxygen from water) and one of the oxygen type from the PEG (1). The 10 first pico seconds are disregarded. Then, once the force is applied to the PEG, a second *fix ave/time* is used.

```
compute myRDF_PEG_H2O all rdf 200 1 8 2 8 cutoff 10
fix myat2 all ave/time 10 4000 50000 c_myRDF_PEG_H2O[*] &
    file PEG-H2O-initial.dat mode vector
```

Add salt to the mixture

You can download the input, data, and parm files I wrote.

It is important to make space for the two salt atoms by modifying the data file as follow:

```
(...)
11 atom types
(...)
```

Additional *mass* and *pair_coeff* lines must also be added to the parm file (be careful to use the appropriate units):

```
(...)
mass 10 22.98 # Na
mass 11 35.453 # Cl
(...)
pair_coeff 10 10 0.04690 2.43 # Na
pair_coeff 11 11 0.1500 4.045
(...)
```

Finally, here I choose to add the ions using two separate *create_atoms* commands with a very small *overlap* values, followed by an energy minimization.

Note also the presence of the *set* commands to give a net charge to the ions.

Evaluate the deformation of the PEG

You can download the input file I wrote.

The key is to combine the *compute dihedral/local*, which computes the angles of the dihedrals and returns them in a vector, with the *ave/histo* functionalities of LAMMPS:

```
compute mydihe all dihedral/local phi
fix myavehisto all ave/histo 10 2000 30000 0 180 500 c_mydihe &
    file initial.histo mode vector
```

Here I choose to unfix *myavehisto* at the end of the first run, and to re-start it with a different file name during the second phase of the simulation.

Nanosheared electrolyte

Induce a Poiseuille flow

Here the *input* script written during the last part *Imposed shearing* of the tutorial is adapted so that, instead of a shearing induced by the relative motion of the walls, the fluid motion is generated by an additional force applied to both water molecules and ions.

To do so, here are the most important commands used to properly thermalize the system:

```
fix mynve all nve
compute tliq fluid temp/partial 0 1 1
fix myber1 fluid temp/berendsen 300 300 100
fix_modify myber1 temp tliq
compute twall wall temp
fix myber2 wall temp/berendsen 300 300 100
fix_modify myber2 temp twall
```

Here, since walls wont move, they can be thermalized in all 3 directions and there is no need for recentering. Instead, one can keep the walls in place by adding springs to every atom:

```
fix myspring wall spring/self 10.0 xyz
```

Finally, let us apply a force to the fluid group along the x direction:

```
fix myadf fluid addforce 3e-2 0.0 0.0
```

The choice of a force equal to $f = 0.03\,\text{kcal/mol/Å}$ is discussed below.

One can have a look at the velocity profiles. The fluid shows the characteristic parabolic shape of Poiseuille flow in the case of a non-slip solid surface. To obtain smooth looking data, I ran the simulation for a total duration of 1 ns. To lower the duration of the computation, don't hesitate to use a shorter duration like 100 ps.

The fitting of the velocity profile was made using the following Poiseuille equation,

$$v = -\alpha \frac{f\rho}{\eta}\left(\frac{z^2}{2} - \frac{h^2}{8}\right),$$

which can be derived from the Stokes equation $\eta\nabla\mathbf{v} = -\mathbf{f}\rho$ where f is the applied force, ρ is the fluid density, η is the fluid viscosity, and $h = 1.2\,\text{nm}$ is the pore size. A small correction $\alpha = 0.78$ was used. This correction compensates the fact that using bulk density and bulk viscosity is obviously no correct in such nanoconfined pore. More subtle corrections could be applied by correcting both density and viscosity based on independent measurement, but this is beyond the scope of the present exercise.

Choosing the right force

The first and most important technical difficulty of any out-of-equilibrium simulation is to choose the value of the forcing f. If the forcing is too large, the system may not be in a linear

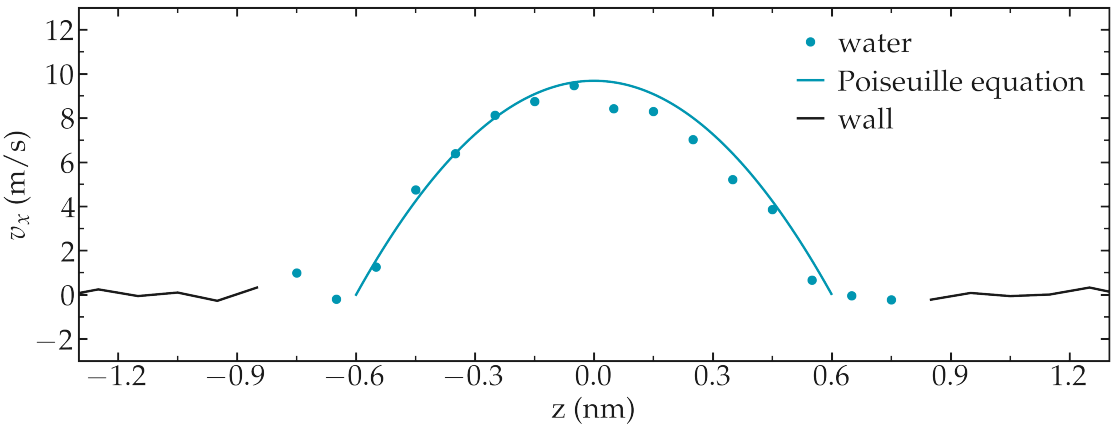


Figure: Velocity profiles of the water molecules along the z axis (disks). The line is the Poiseuille equation.

response regime, meaning that the results are forcing-dependent (and likely quite meaningless). If the forcing is too small, the motion of the system will be difficult to measure due to the low signal-to-noise ratio.

In the present case, one can perform a calibration by running several simulations with different force values f, and then by plotting the velocity of the center of mass v_{cm} of the fluid as a function of the force. Here, I present the results I have obtained by performing the simulations with different values of the forcing. v_{cm} can be extracted by adding the following command to the *input*:

```
variable vcm_fluid equal vcm(fluid,x)
fix myat1 all ave/time 10 100 1000 v_vcm_fluid file vcm_fluid.dat
```

The results show that as long as the force is lower than about 0.04 kcal/mol/Å, there is reasonable linearity between force and fluid velocity.

Water adsorption in silica

Mixture adsorption

You can download the input for the combine water and CO2 adsorption. One of the first step is to create both type of molecules before starting the GCMC:

```
molecule h2omol H2O.mol
molecule co2mol CO2.mol
create_atoms 0 random 5 456415 NULL &
    mol h2omol 454756 overlap 2.0 maxtry 50
create_atoms 0 random 5 373823 NULL &
    mol co2mol 989812 overlap 2.0 maxtry 50
```

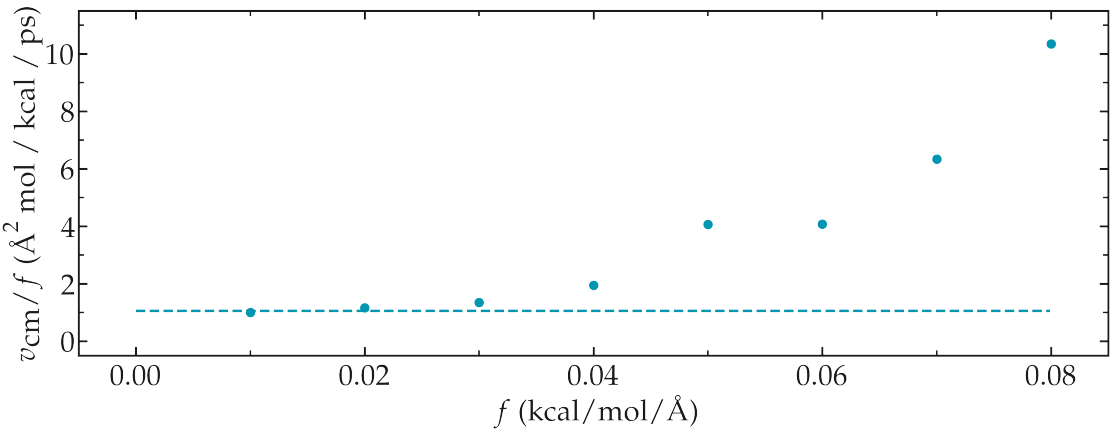


Figure: Ratio between the velocity of the center of mass v_{cm} of the fluid and the forcing f as a function of the forcing

One must be careful to properly write the parameters of the system, with all the proper cross coefficients:

```
pair_coeff * * vashishta ../../Potential/SiO.1990.vashishta &
    Si O NULL NULL NULL NULL
pair_coeff * * lj/cut/tip4p/long 0 0
pair_coeff 1 3 lj/cut/tip4p/long 0.0057 4.42
pair_coeff 1 5 lj/cut/tip4p/long 0.01096 3.158
pair_coeff 1 6 lj/cut/tip4p/long 0.007315 3.2507
pair_coeff 2 3 lj/cut/tip4p/long 0.0043 3.12
pair_coeff 2 5 lj/cut/tip4p/long 0.0101 2.858
pair_coeff 2 6 lj/cut/tip4p/long 0.0065 2.9512
pair_coeff 3 3 lj/cut/tip4p/long 0.008 3.1589
pair_coeff 3 5 lj/cut/tip4p/long 0.01295 2.8924
pair_coeff 3 6 lj/cut/tip4p/long 0.0093 2.985
pair_coeff 4 4 lj/cut/tip4p/long 0.0 0.0
pair_coeff 5 5 lj/cut/tip4p/long 0.0179 2.625854
pair_coeff 6 6 lj/cut/tip4p/long 0.0106 2.8114421
```

Here, I choose to thermalize all species separately:

```
compute ctH2O H2O temp
compute_modify ctH2O dynamic yes
fix mynvt1 H2O nvt temp 300 300 0.1
fix_modify mynvt1 temp ctH2O

compute ctCO2 CO2 temp
compute_modify ctCO2 dynamic yes
fix mynvt2 CO2 nvt temp 300 300 0.1
fix_modify mynvt2 temp ctCO2

compute ctSiO SiO temp
fix mynvt3 SiO nvt temp 300 300 0.1
fix_modify mynvt3 temp ctSiO
```

Finally, adsorption is made with two separates *fix gcmc* commands placed in a loop:

```
label loop
variable a loop 30

fix fgcmc_H2O H2O gcmc 100 100 0 0 65899 300 -0.5 0.1 &
    mol h2omol tfac_insert ${tfac} group H2O shake shak &
    full_energy pressure 100 region system
run 500
unfix fgcmc_H2O

fix fgcmc_CO2 CO2 gcmc 100 100 0 0 87787 300 -0.5 0.1 &
    mol co2mol tfac_insert ${tfac} group CO2 &
    full_energy pressure 100 region system
run 500
unfix fgcmc_CO2

next a
jump SELF loop
```

Here I choose to apply the first *fix gcmc* for the H_2O for 500 steps, then unfix it before starting the second *fix gcmc* for the CO_2 for 500 steps as well. Then, thanks to the *jump*, these two fixes are applied successively 30 times each, allowing for the progressive adsorption of both species.

Adsorb water in ZIF-8 nanopores

You can download the input for the water adsorption in Zif-8, which you have to place in the same folder as the *zif-8.data*, *parm.lammps*, and *water.mol* files.

Apart from the parameters and topology, the *input* is quite similar to the one developped in the case of the crack silica.

You should observe an increase of the number of molecule with time. Run much longer simulation if you want to saturate the porous material with water.

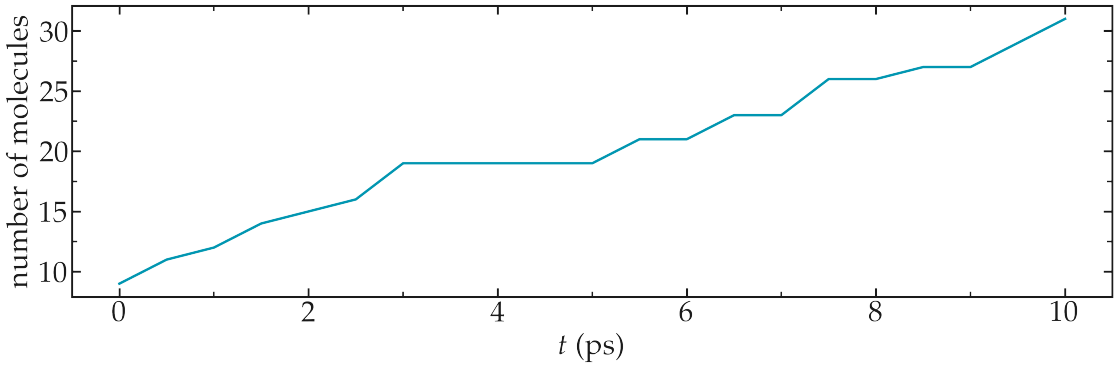


Figure: Number of water molecule in Zif-8 during the first $10\,ps$.

Free energy calculation

The binary fluid that wont mix

You can download the input here.

The solution chosen here was to create two groups (*t1* and *t2*) and apply the two potentials *U1* and *U2* to each group, respectively.

To to so, two separate *fix addforce* are used:

```
group t1 type 1
variable U1 atom ${U0}*atan((x+${x0})/${dlt}) &
    -${U0}*atan((x-${x0})/${dlt})
variable F1 atom ${U0}/((x-${x0})^2/${dlt}^2+1)/${dlt} &
    -${U0}/((x+${x0})^2/${dlt}^2+1)/${dlt}
fix myadf1 t1 addforce v_F1 0.0 0.0 energy v_U1
fix_modify myadf1 energy yes

group t2 type 2
variable U2 atom -${U0}*atan((x+${x0})/${dlt}) &
    +${U0}*atan((x-${x0})/${dlt})
variable F2 atom -${U0}/((x-${x0})^2/${dlt}^2+1)/${dlt} &
    +${U0}/((x+${x0})^2/${dlt}^2+1)/${dlt}
fix myadf2 t2 addforce v_F2 0.0 0.0 energy v_U2
fix_modify myadf2 energy yes
```

60 particles of each type are created, with both types having the exact same properties:

```
mass * 39.95
pair_coeff * * ${epsilon} ${sigma}
```

Feel free to insert some size or mass asymmetry in the mixture, and test how/if it impacts the final potential.

Particles under convection

Add a forcing to all the particles using:

```
fix myconv all addforce 2e-6 0 0
```

It is crutial to choose a forcing that is not *too large*, or the simulation may crash. A forcing that is *too weak* wont have any effect on the PMF.

One can see from the result that the measured potential is tilted, which is a consequence of the additional force that makes it easier for the particles to cross the potential in one of the direction. The barrer is also reduced compared to the case in absence of additional forcing.

Surface adsorption of a molecule

You can download the input here.

Reactive silicon dioxide

Add O2 molecules

In a separate folder, create a new input file, and copy the same first lines as previously in it (just adapt the path to *silica-deformed.data* accordingly):

```
units real
atom_style full

read_data ../../Deform/silica-deformed.data

mass 1 28.0855 # Si
mass 2 15.999 # O

pair_style reaxff NULL safezone 3.0 mincap 150
pair_coeff * * ../RelaxSilica/reaxCHOFe.ff Si O
fix myqeq all qeq/reaxff 1 0.0 10.0 1.0e-6 reaxff maxiter 400
```

Optionally, let us shift the structure to recenter it in the box. The best value for the shift may be different in your case. This step is not necessary, but the recentered system looks better.

```
displace_atoms all move -13 0 0 units box
```

Then, let us import the molecule template *O2.mol* and create 10 molecules. The overlap and maxtry keywords allow us to prevent overlapping between the atoms:

```
molecule O2mol O2.mol
create_atoms 0 random 10 456415 NULL &
    mol O2mol 454756 overlap 3.0 maxtry 50
```

Use the following molecule template named *O2.mol*:

```
2 atoms

Coords

1 -0.6 0 0
2 0.6 0 0

Types

1 2
2 2

Charges

1 0.0
2 0.0
```

The value of 3 Angstroms for the minimum interatomic overlapping is very safe for the present system. Smaller values may lead to molecules being too close from each others.

Finally, let us minimize the energy of the system, and run for 10 ps:

```
minimize 1.0e-4 1.0e-6 100 1000
reset_timestep 0

group grpSi type 1
group grpO type 2
variable totqSi equal charge(grpSi)
variable totqO equal charge(grpO)
variable nSi equal count(grpSi)
variable nO equal count(grpO)
variable qSi equal v_totqSi/${nSi}
variable qO equal v_totqO/${nO}

dump dmp all custom 100 dump.lammpstrj id type q x y z
thermo 5
thermo_style custom step temp etotal press vol v_qSi v_qO
fix myspec all reaxff/species 5 1 5 species.log element Si O

fix mynvt all nvt temp 300.0 300.0 100
timestep 0.5

run 20000
```

You can vizualise the O_2 molecules with VMD, or have a look at the *species.log* file:

```
#  Timestep    No_Moles    No_Specs   Si192O384          O2
          5          11           2           1          10
```

One can see that some reactions occur in the system, and that eventually some of the O_2 molecules react and reabsorb on the main structure:

```
#  Timestep    No_Moles    No_Specs   Si192O388          O2
      20000           9           2           1           8
```

Decorate dandling oxygens

Space must be made for the hydrogen atoms. Modify the *silica-deformed.data* file so that it starts with:

```
576 atoms
3 atom types
```

Also add the mass of the hydrogen:

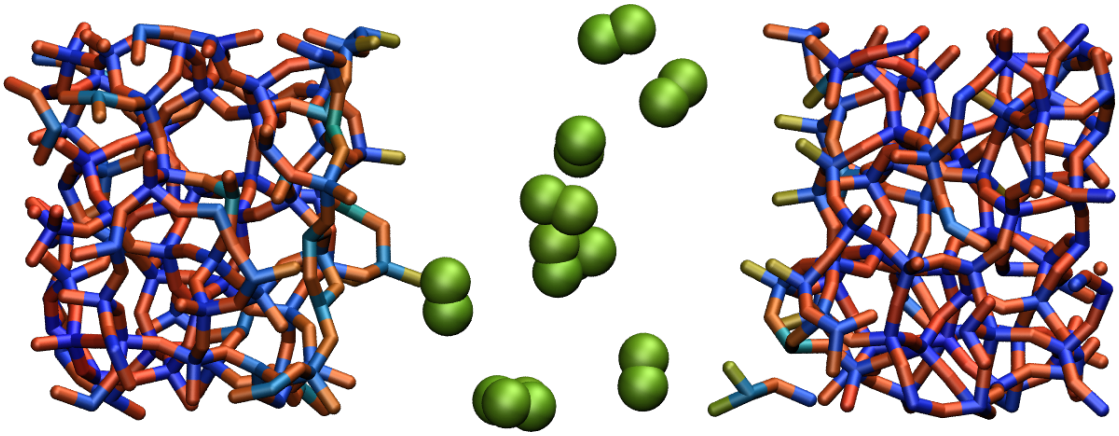


Figure: Deformed structure with some O_2 molecules

```
Masses

1 28.0855
2 15.999
3 1.008
```

It is also important to change the *pair_coeff*:

```
pair_coeff * * ../../RelaxSilica/reaxCHOFe.ff Si O H
```

One can create randomly a few hydrogen atoms:

```
create_atoms 3 random 10 456415 NULL overlap 3.0 maxtry 50
```

Equilibrate the system, you should see the hydrogen atoms progressively decorating the surface of the SiO2 structure:

```
#  Timestep    No_Moles    No_Specs    Si192O384        H
          5          11           2            1       10
(...)
#  Timestep    No_Moles    No_Specs Si192O384H10
       5000           1           1            1
```

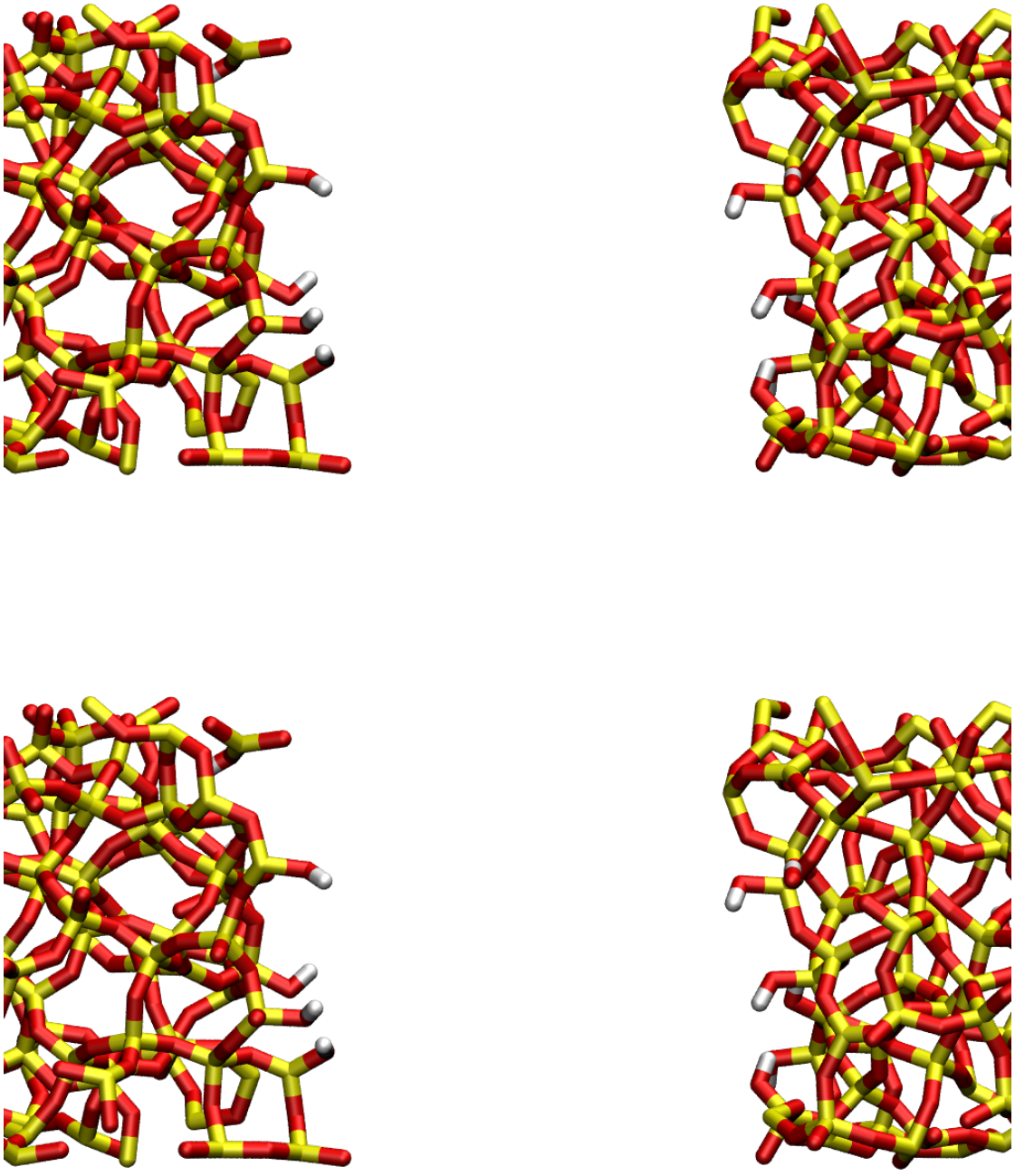
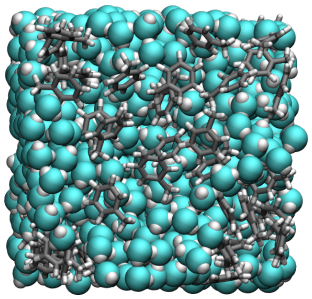


Figure: Hydrogen atoms are in white, oxygen in red, and silicon in yellow.

VMD tutorial

Generate good looking images and movies with VMD



Visual Molecular Dynamics (VMD) is a free molecular graphics software that can be used to visualize molecular dynamics systems. VMD has been used to generate all images of molecular systems here.

The goal of this extra tutorial is to provide some tips to make good looking pictures and videos of molecular systems.

Practical example

To follow this tutorial, download this LAMMPS trajectory file, which corresponds to a mixture of water and toluene.

The water molecules use *types* 1 and 2, and the toluene molecules use *types* 3, 4, and 5.

With Ubuntu/Linux, the *lammptrj* file can be opened with VMD by typing in a terminal:

```
vmd dump.lammpstrj
```

Otherwise, simply open VMD and import the *dump.lammpstrj* file manually using *File -¿ New molecule*.

Go to *Display*, change the view to *Orthographic*, and unselect *Depth Cueing*.

Still in *Display*, select *Axes -¿ Off*.

The representation

In the main windows of VMD, go to *Graphics, Representations*. Within the *Selected Atoms* windows, replace *all* by *type 1*. Here, *type 1* corresponds to the oxygen of the water molecule. Change the *Drawing Method* from *Lines* to *VDW*. Tune the *Sphere Scale* to 0.9, and increase the resolution to 52.

Click on *Create Rep* to create a second representation for the hydrogen of water, select *type 2*, and change the *Sphere Scale* to 0.5.

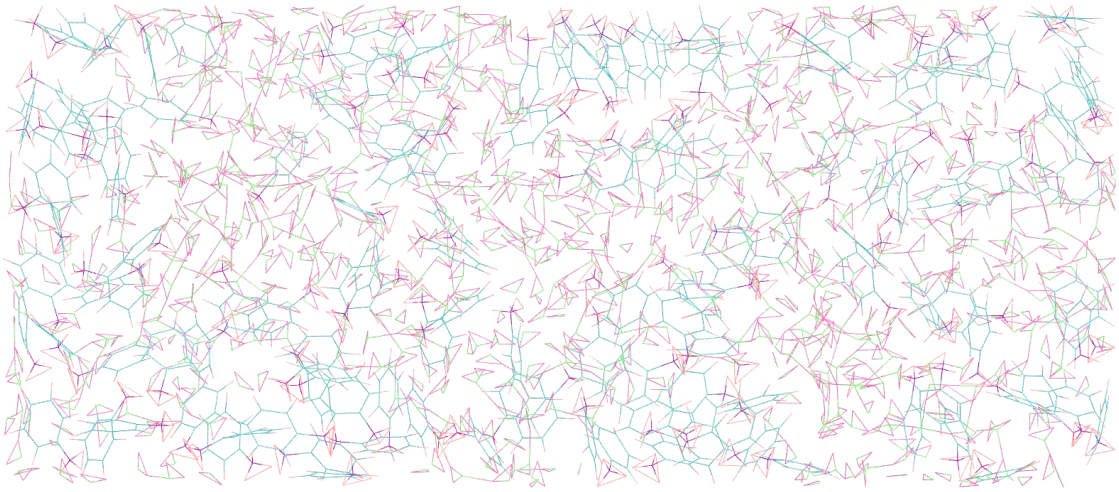


Figure: Initial system in absence of depth cueing and with orthographic view.

Create a third representation for *type 3 4 5*, i.e. all 3 atom types of toluene, respectively carbon, hydrogen, and another carbon.

Choose *DynamicBonds* and increase the *bond resolution* to 52. With *DynamicBonds*, the ends of the bonds are rough. To smooth out the representation, create the fourth and last representation (*VDW* with *Sphere Scale* 0.2) for *types* 3 4 5.

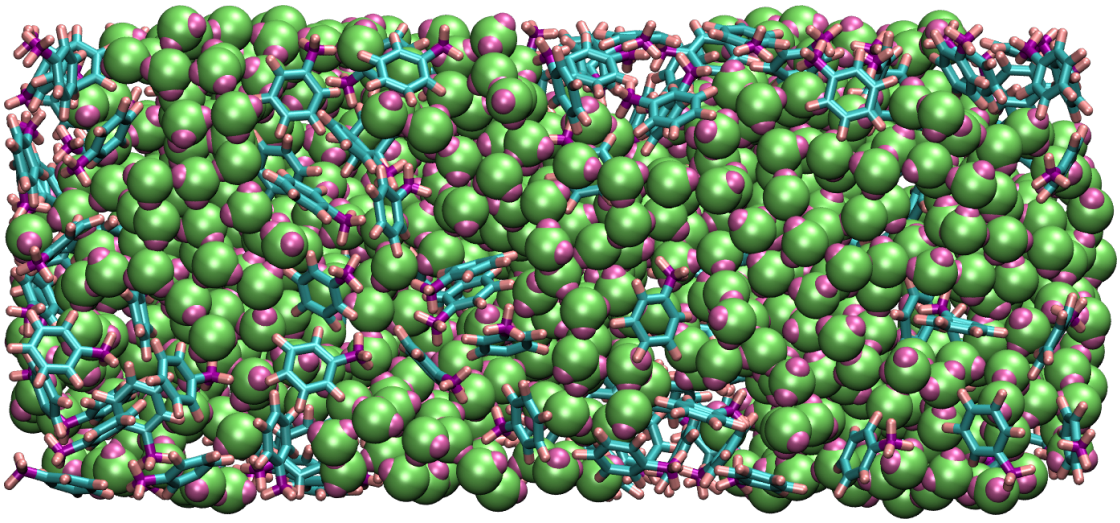


Figure: Orthographic view of the system with improved representation.

The colors

To change the colors, go to *Graphics, Colors*, click on *Display*, then *Background*, and choose the color you prefer (white is better for publication, black can be good looking on presentation with dark background).

Still in the *Color Controls* windows, in *Categories*, click *Name*. In the *Names* sub windows choose *3* (carbon) and select the color silver. Then, do the same for 5 (also a carbon → silver), 4 (hydrogen → white), 2 (hydrogen → white), 1 (oxygen → cyan).

Note that the cyan color is not standard for oxygen. Feel free to change it based on your taste.

Let us slightly change the original *cyan* of VMD by entering manually the values 0.3, 1.0 and 1.0 in the RGB box.

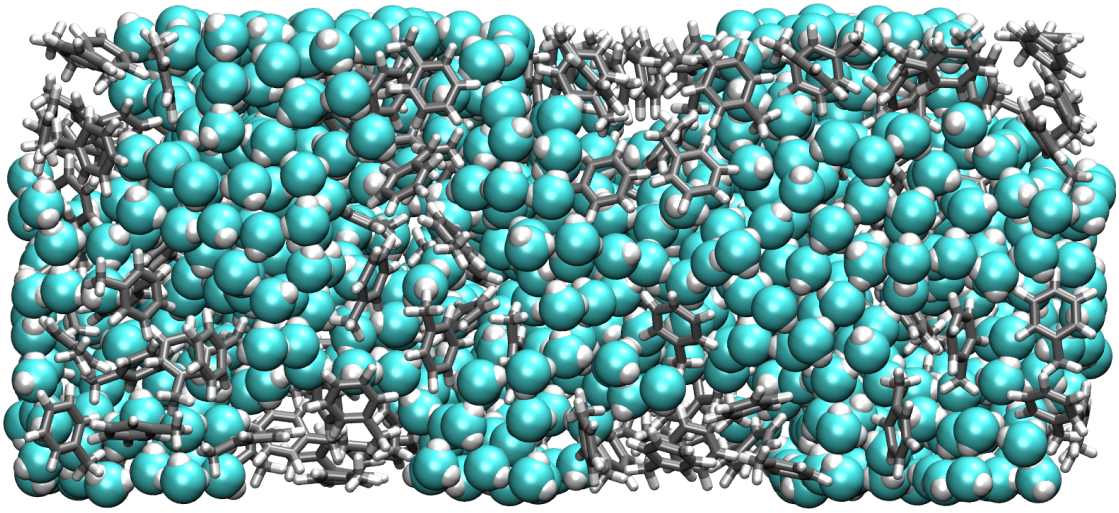


Figure: Orthographic view with improved representation and color.

The materials

In the *Representations* windows, you can choose among several materials that are more or less shiny or opaque.

Let us select the default material named *Opaque*, and change *Diffuse*, *Specular*, and *Shininess*, to 0.56, 0.12, and 0.29, respectively.

Additional options and rendering

Transparent field

A great representation offered by VMD are the *Quick surf*, that can be combined with *transparent* material.

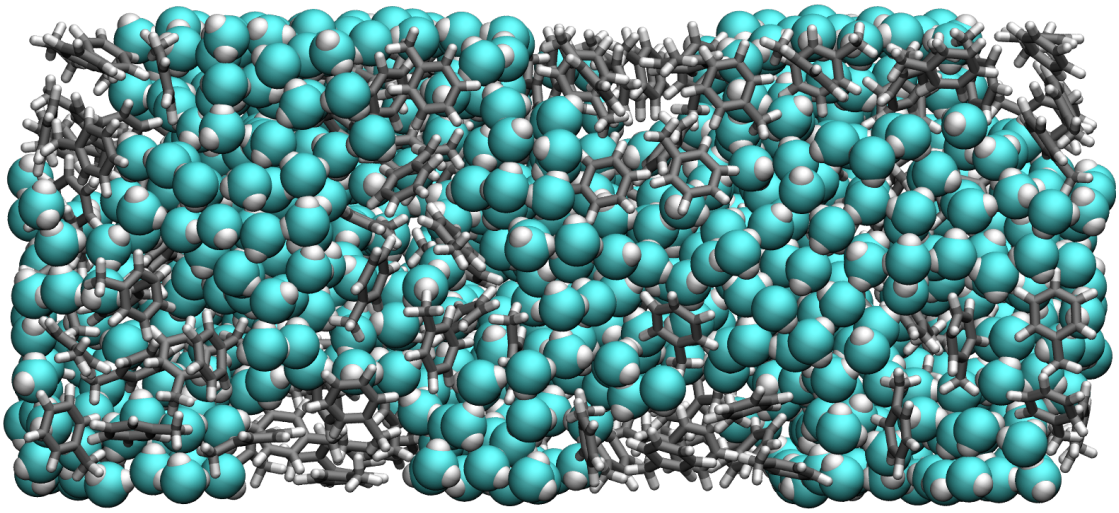


Figure: Orthographic view with improved representation, color, and material. See the corresponding video.

Here I turned off *Light 0*, and turned on all three other defaults lights.

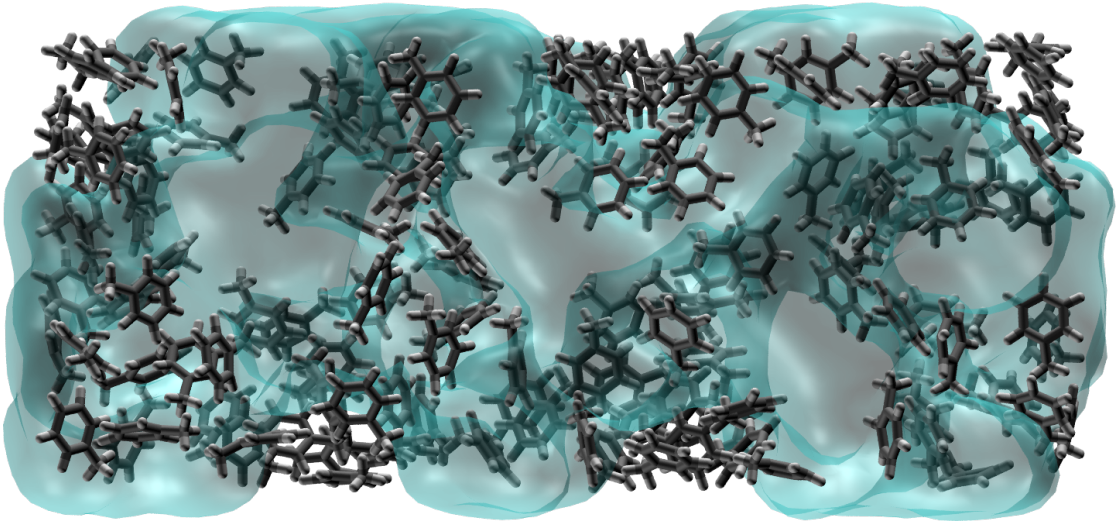


Figure: System with water represented as a transparent field.

Goodsell

VMD also offers the Goodsell's representation, which can be an alternative alternative.

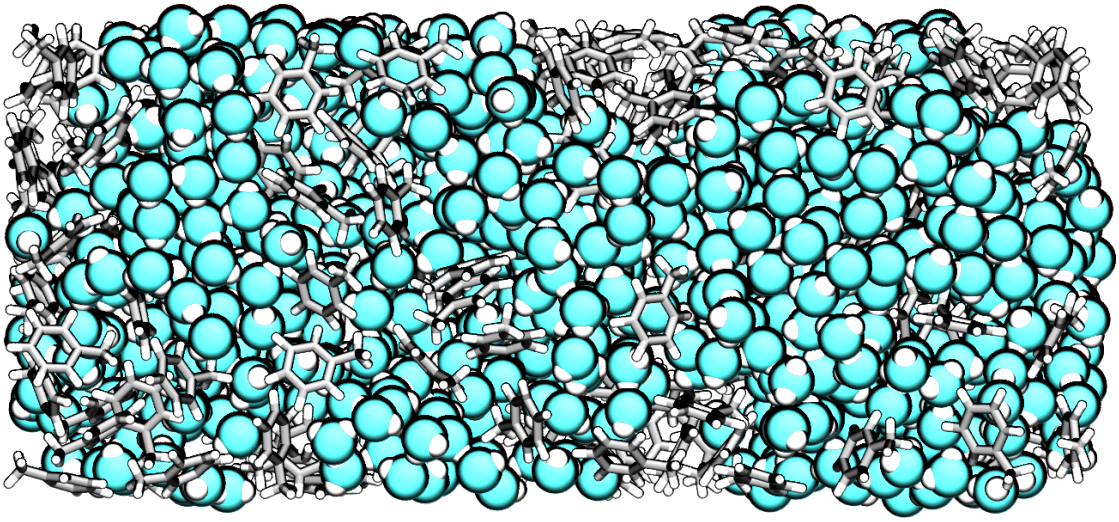


Figure: System in the style of David Goodsell.

Box border

Optionally, you can visualize the borders of the simulation box by typing in the VMD terminal:

```
pbc box -center origin -color black -width 2
```

Saving a state

To avoid redoing all these steps every time VMD is re-opened, one can save the VMD state by clicking *File* $\rightarrow$ *Save vizualisation state*. This state can then be used simply by clicking *File* $\rightarrow$ *Load vizualisation state*.

Rendering image

To generate high a resolution image, go in *File* $\rightarrow$ *Render*, choose *Tachyon*, hit *Start Rendering*.

Rendering movie

To generate a high resolution movie, go in *Extension, Vizualisation*, and *Movie Maker*.

If you hit *Make Movie* directly, the movie generated by VMD will be of poor quality. Instead, it is better to generate a sequence of high resolution images, and assemble these images.

Go in *Movie Settings*, hit *Trajectory* (so the movie will show the system evolving in time, and not rotating on itself), Uncheck *Delete image files*. In *Rendered*, choose *Tachyon*, then hit *Make Movie*.

From the linux terminal (not the VMD terminal), assemble the images (all starting with *untitled*) into a single movie by typing:

```
ffmpeg -r 60 -i untitled.%05d.ppm -vcodec libx264 \
-crf 0  -pix_fmt yuv420p myvideo.mp4
```

You may receive the following error:

```
width not divisible by 2 (1363x1134)
```

In that case, simply remove one line of pixel with the command:

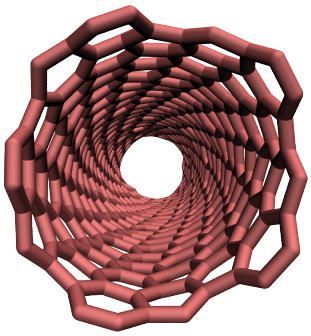
```
for file in untitled.*.ppm; do convert $file -crop 1362x1134+0+0 $file; done
```

To convert the video in *webp*, for web integration, use:

```
ffmpeg -i myvideo.mp4 -vcodec libwebp -filter:v fps=fps=20 \
-lossless 1 -loop 0 -preset default -an -vsync 0 myvideo.webp
```

MDAnalysis tutorial

Perform post-mortem analysis using Python



There are two main ways to analyze data from a MD simulation: (1) on-the-fly analysis, like is done for instance using *fix ave/time*, and (2) post-mortem analysis. Post-mortem analysis can be performed using the atom coordinates saved in the *lammpstrj* file.

The goal of this extra tutorial is to provide some tips to import *lammpstrj* trajectory file into Python.

Counting the bonds of a CNT

Here, we re-use the trajectory generated during the second part *Breakable bonds* of the Pulling on a carbon nanotube tutorial. It is recommended that you follow this tutorial first, but you can also directly download the dump file and the data file and continue with this MDA tutorial.

Create a Universe

Open a new Jupyter notebook and call it *measure_bond_evolution.ipynb*. First, let us import both *MDAnalysis* and *NumPy* by copying the following lines into *measure_bond_evolution.ipynb*.

```
import MDAnalysis as mda
import numpy as np
```

Then, let us create a MDAnalysis *universe* using the LAMMPS data file *cnt_atom.data* as topology, and the *lammpstrj* file as trajectory. Add the following lines into *measure_bond_evolution.ipynb*:

```
path_to_data = "./"
u = mda.Universe(path_to_data + "cnt_deformed.data",
                 path_to_data + "dump.lammpstrj",
                 topology_format="data", format="lammpsdump",
                 atom_style='id type xs ys zs',
                 guess_bonds=True, vdwradii={'1':1.7})
```

Since the *.data* file does not contain any bond information the original bonds are guessed using the bond guesser of MDAnalysis using *guess_bonds=True*.

Note that the bond guesser of MDAnalysis will not update the list of bond over time, so we will need to use a few tricks to extract the evolution of the number of bond with time.

Let us create a single atom group named *cnt* and containing all the carbon atoms, i.e. all the atoms of type 1, by adding the following lines into *measure_bond_evolution.ipynb*.

```
cnt = u.select_atoms("type 1")
```

Some basics of MDAnalysis

MDAnalysis allows us to easily access information concerning the simulation, such as the number of atoms, or the number of frames in the trajectory:

```
print("Number of atoms =", cnt.n_atoms)
print("Number of frames =", u.trajectory.n_frames)

Number of atoms = 690
Number of frames = 286
```

It is also possible to access the indexes of the atoms that are considered as bonded by the bond guesser of MDAnalysis:

```
print(cnt.atoms.bonds.indices)

[[  0   2]
[  0  23]
[  0  56]
(...)
[686 687]
[686 689]
[688 689]]
```

MDAnalysis also offer the possibility to loop over all the frame of the trajectory using:

```
for ts in u.trajectory:
    print(ts.frame)

0
1
2
3
(...)
283
284
285
```

The positions of the atoms can also be obtained using:

```
u.atoms.positions

array([[ 75.14728 ,  78.17872 ,  95.61408 ],
[ 75.33008 ,  77.751114,  93.20232 ],
[ 75.550476,  77.34152 ,  94.54224 ],
...,
[ 84.66992 ,  82.24888 , 143.84988 ],
[ 84.66992 ,  82.24888 , 147.60156 ],
[ 84.85272 ,  81.82128 , 146.26175 ]], dtype=float32)
```

where the three columns of the array are the x, y, and z coordinates of the atoms.

Counting the bonds

In order to measure the evolution of the number of bonds over time, let us loop over the trajectory and manually extract the inter-atomic distance over time.

To to so, for every step of the trajectory, let us loop over the indexes of the atoms that were initially detected as bonded, and calculate the distance between the two atoms, which can be done using:

```
for ts in u.trajectory:
    for id1, id2 in cnt.atoms.bonds.indices:
        # detect positions
        pos1 = u.atoms.positions[u.atoms.indices == id1][0]
        pos2 = u.atoms.positions[u.atoms.indices == id2][0]
        r = np.sqrt(np.sum((pos1-pos2)**2))
```

Then, let us assume that if r is larger that a certain cut-off value of, let's say, $1.8\ \mathring{A}$, the bond is broken:

```
for ts in u.trajectory:
    for id1, id2 in cnt.atoms.bonds.indices:
        pos1 = u.atoms.positions[u.atoms.indices == id1][0]
        pos2 = u.atoms.positions[u.atoms.indices == id2][0]
        r = np.sqrt(np.sum((pos1-pos2)**2))
        if r < 1.8:
            print("the bond has a length", r, "Å")
        else:
            print("the bond is broken")
```

Finally, let us store both mean length of bonds and total number of bond in lists.

```
lbond_vs_frame = []
nbond_vs_frame = []
for ts in u.trajectory:
    frame = ts.frame
    all_bonds_ts = [] # temporary list to store bond length
    for id1, id2 in cnt.atoms.bonds.indices:
        pos1 = u.atoms.positions[u.atoms.indices == id1]
        pos2 = u.atoms.positions[u.atoms.indices == id2]
        r = np.sqrt(np.sum((pos1-pos2)**2))
        if r < 1.8:
            all_bonds_ts.append(r)
    mean_length_bonds = np.mean(all_bonds_ts)
    number_of_bond = len(all_bonds_ts)/2 # divide by 2 to avoid counting twice
    lbond_vs_frame.append([frame, mean_length_bonds])
    nbond_vs_frame.append([frame, number_of_bond])
```

The data can then be saved to files:

```
np.savetxt("number_bond_vs_time.dat", nbond_vs_time)
np.savetxt("length_bond_vs_time.dat", lbond_vs_time)
```

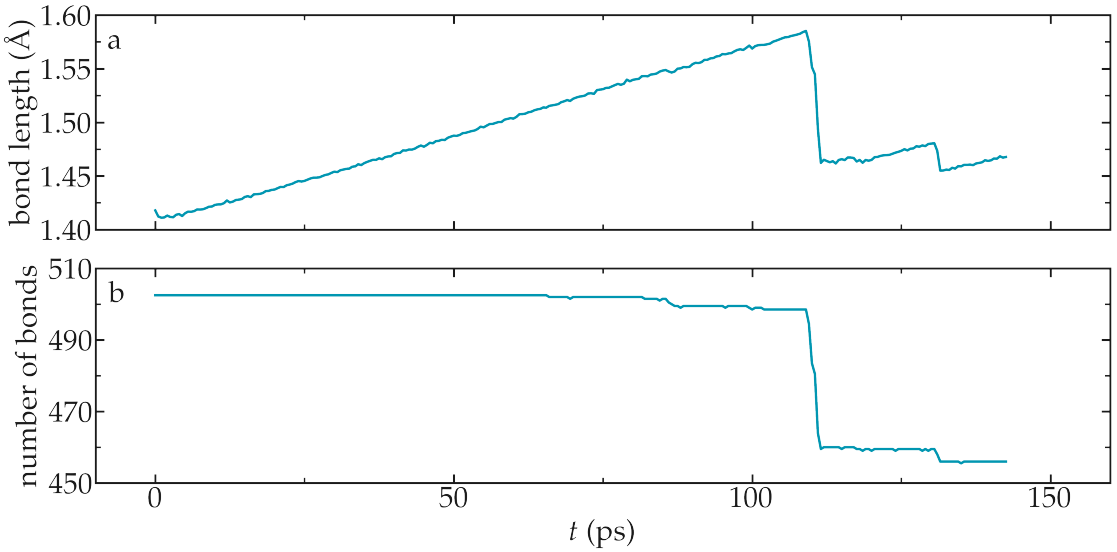


Figure: Evolution of the average bond length (a) and bond number (b) as a function of time.

Bond length distributions

Using a similar script, let us extract the bond length distribution at the beginning of the simulation (let us say the 20 first frame), as well as near the maximum deformation of the CNT:

```
for ts in u.trajectory:
    all_bonds_ts = []
    for id1, id2 in cnt.atoms.bonds.indices:
        pos1 = u.atoms.positions[u.atoms.indices == id1]
        pos2 = u.atoms.positions[u.atoms.indices == id2]
        r = np.sqrt(np.sum((pos1-pos2)**2))
        if r < 1.8:
            all_bonds_ts.append(r)
    if frame > 0: # ignore the first frame
        histo, r_val = np.histogram(all_bonds_ts, bins=50, range=(1.3, 1.65))
        r_val = (r_val[1:]+r_val[:-1])/2
        bond_length_distributions.append(np.vstack([r_val, histo]))
```

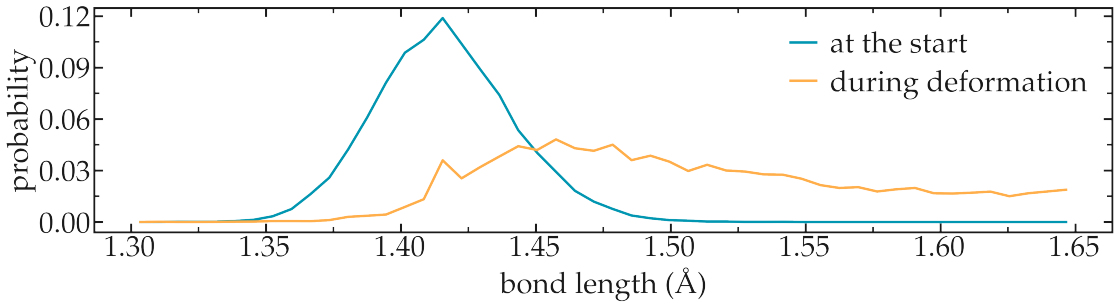


Figure: Distribution in bond length near the start of the simulation, as well as near the maximum deformation of the CNT.