A Set of Tutorials for the LAMMPS Simulation Package [Article v1.0]

Simon Gravelle[1*]

[1] University Grenoble Alpes, CNRS, LIPhy, Grenoble, 38000, France

This LiveCoMS document is maintained online on GitHub at https://github.com/lammpstutorials/lammpstutorials-article; to provide feedback, suggestions, or help improve it, please visit the GitHub repository and participate via the issue tracker.

This version dated July 10, 2024

Abstract The rise of extensive open-source molecular simulation codes allows users to launch simulations without implementing inter-particle interactions or integrating Newton's equations of motion. However, while performing molecular simulations is now accessible to all, the wide array of options and potential pitfalls in these open-source tools can complicate their use, particularly for beginners. LAMMPS is a versatile molecular simulation code designed for modeling particles in a broad range of materials science and computational chemistry applications. It is widely used for its ability to run simulations on large-scale parallel computing environments, making it ideal for complex and extensive simulations of atomic and molecular systems. This article introduces seven tutorials designed to make learning LAMMPS more accessible to new users. The first four tutorials cover the basics of running molecular simulations in LAMMPS with systems of varying complexities, including a simple fluid and a carbon nanotube. The last three tutorials address more advanced molecular simulations, specifically the use of a reactive force field, umbrella sampling, and grand canonical Monte Carlo.

***For correspondence:**
simon.gravelle@cnrs.fr (SG)

1 Introduction

Molecular simulations (MS) can be used to model a large variety of atomic and coarse-grained systems, including solids, fluids, polymers, and biomolecules, as well as complex interfaces and multicomponent systems. Various MS methods exist, molecular dynamics (MD) and Monte Carlo (MC) being among the most commonly used. MD is the preferred method for obtaining the accurate dynamics of a system, as it relies on solving Newton's equations of motion. For systems with many degrees of freedom or complex energy landscapes, the MC method can be a better choice than MD because it allows for efficiently exploring a configuration space. MC involves performing random changes to the system configuration that are either accepted or rejected based on energy criteria [1, 2]. MS allow for measuring a broad variety of properties, including structural properties (e.g. bond length, inter-atomic distances), thermodynamic properties (e.g. temperature, pressure, volume), dynamic behaviors (e.g. diffusion coefficient, viscosity), and mechanical responses (e.g. elastic constant, Poisson ratio). Some of these quantities can be directly compared with experimental data, enabling the validation of the simulated system, while others, available only through simulations, are often useful for interpreting experimental data [3].

LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) is a highly flexible open-source MS tool [4]. Over the years, a broad variety of particle interaction models have been implemented in LAMMPS, enabling it to model, among others, atomic, polymeric, biological, metallic, granular, and coarse-grained systems. LAMMPS can be used on a single CPU core or a supercomputing system, and it can handle complex and large-scale simulations efficiently. However, LAMMPS requires users to provide detailed input

files that can be complex to write, particularly given the overwhelming number of options offered by LAMMPS, all contained in extensive documentation that can sometimes be difficult for new users to navigate. In addition to the intrinsic complexity of LAMMPS, performing accurate MS involves making several complex and system-specific decisions, such as selecting the thermodynamic ensemble (e.g. micro-canonical, grand-canonical), determining the simulation duration, and choosing the sets of parameters describing the interactions between the atoms (the so-called force field) [5]. The tutorials presented in this article aim to shorten the learning curve and help users perform accurate and reliable MS using LAMMPS.

1.1 Scope

This set of tutorials consists of seven tutorials arranged in order of increasing difficulty. The novelties associated with each tutorial are briefly described below.

In tutorial 1, the structure of LAMMPS input files is illustrated through the creation of a simple Lennard-Jones fluid system. Basic LAMMPS commands are used to set the interaction between the atoms, perform an energy minimization, and finally a simple MD simulation in the canonical (NVT) ensemble. This tutorial also illustrates the use of the LAMMPS graphical user interface (GUI).

In tutorial 2, a more complex system where atoms are connected by bonds is used: a small carbon nanotube. The use of both classical and reactive force fields (here AIREBO) is illustrated. An external deformation is applied to the CNT, and its deformation is measured. This tutorial illustrates the use of the YAML format to import the data from *log* file into Python.

In tutorial 3, two components, liquid water and a polymer molecule, are merged and equilibrated. A long-range solver is used to handle the electrostatic interactions, and the system is equilibrated in the isothermal-isobaric (NPT) ensemble. A stretching force is then applied to the polymer.

In tutorial 4, a fluid confined by two walls is simulated, illustrating the specifics of simulating systems with fluid-solid interfaces. This tutorial involves a slightly more complex water model than tutorial 3: the four-point TIP4P model. In this tutorial, non-equilibrium MD is performed through the imposed shearing of the fluid by the moving walls.

In tutorial 5, the reactive force field called ReaxFF is used in conjunction with the charge equilibration method QEq. This tutorial describes a situation where the partial charges of the atoms adapt to their local environment.

In tutorial 6, a Monte Carlo simulation in the Grand Canonical ensemble is implemented to illustrate the use of LAMMPS for simulating an open system that can exchange particles with a reservoir.

In tutorial 7, an advanced free energy method named umbrella sampling is implemented. Through the calculation of an energy barrier, this tutorial describes a protocol that can be used to deal with energy landscapes that prevent their sampling through classical MD or MC methods.

2 Prerequisites

2.1 Background knowledge

This set of tutorials requires no prior knowledge of LAMMPS. The tutorials assume that the user is familiar with the use of a plain-text editor such as VIM or Emacs, as well as with the most basic Unix/Linux command-line interactions. In addition, prior knowledge of the basics of molecular simulations and statistical physics would be beneficial. The user may refer to *Understanding molecular simulation* by Daan Frenkel and Berend Smit [1], as well as *Computer simulation of liquids* by Michael Allen and Dominic Tildesley [2]. Additionally, to better understand the fundamental concepts behind the soft matter systems simulated in these tutorials, the user can refer to *Basic concepts for simple and complex liquids* by Jean-Louis Barrat and Jean-Pierre Hansen [6], as well as *Theory of simple liquids: with applications to soft matter* by Jean-Pierre Hansen and Ian Ranald McDonald [7].

2.2 Software/system requirements

The LAMMPS version 2Aug2023 is required to follow the tutorials [4]. Install it by following the instructions of the LAMMPS website, or download directly a pre-compiled LAMMPS package (docs.lammps.org). Optionally, to follow Tutorial 1, choose a package that was compiled with the LAMMPS-GUI. Depending on your operating 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

To write and edit the LAMMPS input files, a text editor such as VIM or Emacs is required. Additionally, to visualize the trajectories of the atoms, VMD version 1.9.3 will be used [8]. Feel free to use an alternative visualization software like Ovito. Finally, a graph plotting tool is required. To plot the results from the simulations, Matplotlib Pyplot version 3.5.2 is used together with Python version 3.11.4 [9, 10], but any plotting tool can be used, such as XmGrace or Gnuplot.

Figure 1. The binary mixture simulated during Tutorial 1. The atoms of type 1 are represented as small green spheres and the atoms of type 2 as large blue spheres.

3 Content and links

The tutorials described in this article can be accessed at lammpstutorials.github.io, where additional exercises with solutions are also provided. All files and inputs required to follow the tutorials are available from a dedicated GitHub account, github.com/lammpstutorials. In the following, all the commands that must be copied and pasted into LAMMPS input files or the terminal are written in a `verbatim` environment, and the file and folder names are written in *italic*.

3.1 Tutorial 1: Lennard-Jones fluid

The objective of this tutorial is to perform a simple MD simulation using LAMMPS. The system is a Lennard-Jones fluid composed of neutral particles with two different diameters, contained within a cubic box with periodic boundary conditions (Fig. 1). In this tutorial, the temperature of the system is maintained using a Langevin thermostat [11], and basic quantities are extracted from the system, including the potential and kinetic energies.

For LAMMPS-GUI users

Tutorial 1 can be followed using the LAMMPS-GUI. Instructions for GUI users are written in these boxes that can be ignored by non-GUI users.

3.1.1 My first input

To run a simulation using LAMMPS, one needs to write a series of commands in an input script. For clarity, the input scripts written for this first tutorial will be divided into five categories which we are going to fill out 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 hash symbol (#) is a comment 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 do not necessarily need to 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 files 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.

For LAMMPS-GUI users

Either open the *input.lammps* file using LAMMPS-GUI, or simply open LAMMPS-GUI. In this later case, an empty buffer appear, in which LAMMPS commands can be written.

System initialization

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 (e.g. periodic or non-periodic),
- the type of atoms (e.g. uncharged single dots or spheres with angular velocities).

Enter the following lines into *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 unit system called *LJ* Lennard-Jones, in which all quantities are unitless. The second line, *dimension 3*, indicates that the simulation is in 3D. The third line, *atom_style atomic*, that the *atomic* style will be used, therefore each atom is just a dot with a mass. 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) [12, 13]:

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

where r is the inter-particle distance, ϵ_{ij} is the depth of potential well that sets the interaction strength, and σ_{ij} is the distance parameter or particle effective size. Here, the indexes

ij refer to the particle types *i* and *j*. 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).

System definition

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 to create different initial conditions for the simulation. The fourth line creates 100 atoms of type 2.

Simulation Settings

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. 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 ϵ_{11} = 1.0 and the distance parameter σ_{11} = 1.0. Similarly, the last line sets the Lennard-Jones coefficients for the interactions between atoms of type 2, ϵ_{22} = 0.5, and σ_{22} = 3.0. By default, LAMMPS calculates the cross coefficients between the different atom types using geometric average: $\epsilon_{ij} = \sqrt{\epsilon_{ii}\epsilon_{jj}}$, $\sigma_{ij} = \sqrt{\sigma_{ii}\sigma_{jj}}$.

Energy minimization

The system is now fully parametrized. Let us fill the two last remaining sections by adding the following lines into *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* command instructs LAMMPS to perform an energy minimization of the system. By default, LAMMPS uses the conjugate gradient (CG) algorithm [14].

Run the simulation by typing the following command in a terminal:

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

where the command *lmp* is linked to a compiled version of LAMMPS.

For LAMMPS-GUI users

Run the simulation by selecting *Run LAMMPS from Editor Buffer* from the *Run* menu.

As the simulation progresses, the potential energy can be seen to decrease from a large positive value to a negative value (see also the orange lines in Fig. 2 a). The initially large and positive value of the potential energy 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 other.

For LAMMPS-GUI users

As the simulation runs, a windows showing the evolution of the temperature over time pops-up. Switch from the temperature to the potential energy by choosing *PotEng* as the *Select data*, or to the kinetic energy by choosing *KinEng*.

Molecular dynamics

The system is now ready. Let us continue by completing the input script and adding commands to perform a molecular

dynamics simulation, starting from the final state of the previous energy minimization step. In the same input script, after the *minimization* command, add the following lines:

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

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 or re-define the system. 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. Then, let us add a second *Run* section:

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

The *fix nve* is used to update the positions and the velocities of the atoms in the group *all* at every step. The group *all* is a default group that contains every atom. The second fix applies a Langevin thermostat to the atoms in the group *all*, with a desired initial temperature of 1.0 (unitless), and a final temperature of 1.0 as well [11]. A *damping* parameter of 0.1 is used. The *damping* parameter determines how rapidly the temperature is relaxed to its desired value. The number *1530917* is a seed, you can change it to perform statistically independent simulations. Finally, the last two lines set the value of the *timestep* and the number of steps for the *run*, respectively, corresponding to a total duration of 50 (unitless).

Run the simulation again using LAMMPS. From the information printed in the terminal, one can see that the temperature *Temp* starts from 0, but rapidly reaches the requested value and stabilizes itself near $T = 1$ (unitless). From what has been printed in the *log* file, one can plot the energy of the system over time. The potential energy, p_e, rapidly decreases during energy minimization (Fig. 2 a). Then, after the molecular dynamics simulation starts, p_e increases until it reaches a plateau value of about -0.25. The kinetic energy, k_e, is equal to zero during energy minimization and then increases during molecular dynamics until it reaches a plateau value of about 1.5 (Fig. 2 b).

Trajectory visualization

To visualize the trajectories of the atoms, we first need to print 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
```

Figure 2. a) Potential energy (p_e) of the binary mixture as a function of the time t. b) Kinetic energy (k_e) as a function of t.

Run the *input.lammps* using LAMMPS again. The file called *dump.lammpstrj* created by LAMMPS within *my-first-input/* can be opened using VMD (The snapshot in Fig. 1 was made using VMD).

For LAMMPS-GUI users

Generate an image of the system by selecting *Create image* from the *Run* menu.

3.1.2 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.

Control the initial atom positions

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

To create the atoms of types 1 and 2 in two separate regions, 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 cylinder_in cylinder z 0 0 10 &
    INF INF side in
region cylinder_out cylinder z 0 0 10 &
    INF INF side out
create_atoms 1 random 1000 341341 cylinder_out
create_atoms 2 random 150 127569 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, compared 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 called *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 minimization step.

Run the *input.min.lammps* script using LAMMPS. As soon as the simulation starts, a new dump file called *dump.min.lammpstrj* must appear in the folder. This *.lammpstrj* can be used to visualize the atom's trajectories during minimization using VMD. At the end of the simulation, a file called *minimized_coordinate.data* is created by LAMMPS. If you open *minimized_coordinate.data* with a text editor, you can see that it contains all the information necessary to restart the simulation, such as the number of atoms, the box size, the *masses*, and the *pair_coeffs*. The *minimized_coordinate.data* file also contains the final positions of the atoms within the *Atoms* section. The first five 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), and the positions of the atoms *x*, *y*, *z*. The last three columns are image flags that keep track of which atoms crossed the periodic boundary.

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 called *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
```

Here, instead of creating a new region and adding atoms to it, we can simply import the previously saved configuration by adding the following command to *input.md.lammps*:

```
# 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 minimization. To start the simulation from a clean slate, 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*:

```
region cylinder_in cylinder z 0 0 10 &
    INF INF side in
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 cylinder_in
group group_region_out 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 are used to create groups containing all the atoms of type 1 and all the atoms of type 2, respectively. 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 the intersection between the

Figure 3. Evolution of the system during mixing. The three snapshots show respectively the system at t = 0 (left panel), t = 75 (middle panel), and t = 1500 (right panel).

previously defined groups. Finally, the two *delete_atoms* commands delete the atoms of type 1 that are located within the cylinder and the atoms of type 2 that are located outside the cylinder, respectively. Add the following lines into *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 n_type1_in &
    equal count(group_type_1,cylinder_in)
variable n_type2_in &
    equal count(group_type_2,cylinder_in)
fix myat1 all ave/time 10 200 2000 &
    v_n_type1_in file population1vstime.dat
fix myat2 all ave/time 10 200 2000 &
    v_n_type2_in file population2vstime.dat
```

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

In addition to counting the atoms in each region, let us also extract the coordination number per atom between atoms of types 1 and 2. The coordination number is a measure of the average number of type 2 atoms in the vicinity of type 1 atoms, which serves as a good indicator of the degree of mixing in a binary mixture. Add the following 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 atoms i-j. Such an array can't be printed by *fix ave/time*. Finally, let us complete the script by adding the following lines into *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 from the previous simulation. First, the *velocity create* command attributes an initial velocity to each 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, which ensures that the total random force applied to the atoms is equal to zero. Run *input.md.lammps* using LAMMPS. Open the *dump.md.lammpstrj* file using VMD to observe the mixing of the two populations over time (see also Fig. 3). From the generated *.dat* file, one can plot the number of atoms in each region as a function of time (Fig. 4 a). In addition, as the mixing progresses, the coordination number between atoms of types 1 and 2 increases from about 0.01 to 0.35 (Fig. 4 b). This indicates that, as time progresses, more and more particles of type 1 get into contact with particles of type 2, as expected during mixing.

3.2 Tutorial 2: Pulling on a carbon nanotube

In this tutorial, the system of interest is a small carbon nanotube (CNT) in an empty box (Fig. 5). An external force is imposed on the CNT, and its deformation is measured over time. To illustrate the difference between classical and reactive force fields, this tutorial is divided into two parts. Within the first part, a classical force field is used and the bonds between the atoms of the CNT are unbreakable. Within the second part, a reactive force field (named AIREBO [15]) is used, allowing for the breaking of chemical bonds when the CNT undergoes strong deformation.

Figure 4. a) Evolution of the number N_{in} of atoms within the *cylinder_in* region as a function of the time t. b) Evolution of the coordination number C_{1-2} between atoms of types 1 and 2.

Figure 5. The carbon nanotube (CNT) simulated during Tutorial 2.

3.2.1 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 to the atoms, the bonds remain intact during the simulation. The bonds between neighbor atoms typically consist of springs with given equilibrium distances r_0 and a constant k_b: $U_b = k_b (r - r_0)^2$. Additionally, angular and dihedral constraints are usually applied to maintain the relative orientations of neighboring atoms.

Download directly the CNT topology *cnt_molecular.data*. It was created using VMD and TopoTools [16], and contains information about the positions of the carbon 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 called *unbreakable-bonds/*.

The LAMMPS input

Create a new text file within *unbreakable-bonds/* and name it *input.lammps*. Copy the following lines into 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 Ångstrom and 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 matter here, as the box boundaries were placed far from the CNT. Just like in the previous tutorial, 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 Ångstroms from each other 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. More details about the potentials used here are given in the LAMMPS documentation. The *special_bonds* command

sets the weighting factors for the Lennard-Jones interaction between atoms directly connected by one bond, two bonds, and three bonds, respectively. The last command, *read_data*, imports the *cnt_molecular.data* file previously generated with VMD, which contains information about the box size, atom positions, etc.

We need to specify the parameters of both bonded and non-bonded potentials. Here, the parameters are taken from the OPLS-AA (Optimised Potentials for Liquid Simulations-All-Atom) force field [17]. Create a new text file in the *unbreakable-bonds/* folder and name it *parm.lammps*. Copy the following lines into it:

```
pair_coeff 1 1 0.066 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 parameters for the non-bonded Lennard-Jones interaction ϵ_{11} = 0.066 kcal/mol and σ_{11} = 3.4 Å for the only type of atom of the simulation; the carbon atom of type 1. The *bond_coeff* provides the equilibrium distance r_0 = 1.4 Å as well as the spring constant k_b = 469 kcal/mol/Å^2 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 θ_0 and constant for the potential between three neighbor 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 into the *input.lammps* file:

```
include parm.lammps
```

Prepare the 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 into *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 moves 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 into *input.lammps*:

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

Such a 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 on 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 into *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 Ångstroms, and z_{min} to the coordinate of the first atoms along z plus 0.5 Ångstroms. Then, three regions are defined and correspond respectively to: $z < z_{min}$, (*rbot*, for region bottom) $z_{min} > z > z_{max}$ (*rmid*, for region middle), and $z > z_{max}$ (*rtop*, for region 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 from the *carbon_top* and *carbon_bot* groups are represented in white in Fig. 6, and the atoms from the *carbon_mid* group in pink.

Figure 6. CNT with the atoms from the *carbon_mid* group in pink, and the atoms from the *carbon_top* and the *carbon_bot* groups in white.

Figure 7. Same CNT as in Fig. 6, but with 10 deleted atoms. The 10 deleted atoms were chosen randomly from the central part of the CNT.

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
```

Finally, to start from a less ideal state and create a system with some defects, let us randomly delete a small fraction of the carbon atoms. To avoid deleting atoms that are too close to the edges, let us define a new region name *rdel* that starts 2 Å 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) (Fig. 7).

The molecular dynamics

Let us specify the thermalization and the dynamics of the system. Add the following lines into *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 atom 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. 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 [18]. 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 the 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.

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). The *velocity set* commands impose the velocity of a group of atoms at the start of a run but do 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 won't 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.

Data extraction

Next, 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 cnt_length.dat
fix at2 all ave/time 10 10 100 f_mysf1[1] &
    f_mysf2[1] file edge_force.dat
```

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

Figure 8. Evolution of the length L of the CNT with time. The CNT starts deforming at t = 5 ps.

Figure 9. Evolution of the total energy E_{tot} of the system with time t. The CNT starts deforming at t = 5 ps. Here, the force field is OPLS-AA (i.e. the bonds are unbreakable).

```
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 temperature_middle_group.dat
```

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 line yaml

timestep 1.0
run 5000
```

With the *thermo_modify* command, we specify to LAMMPS that the temperature T_{mid} must be printed in the terminal instead of the temperature of the entire system. This choice is motivated by the presence of frozen parts with an effective temperature of 0 K, which makes the average temperature of the entire system not relevant. The *thermo_modify* command also imposes the use of the YAML format that can easily be read by Python (see below). Let us impose a constant velocity deformation on 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:

```
velocity carbon_top set 0 0 0.0005
velocity carbon_bot set 0 0 -0.0005
run 10000
```

The chosen velocity for the deformation is 100 m/s, or 0.001 Å/fs.

The length L of the CNT increases linearly over time for t > 5 ps (Fig. 8), as expected from the imposed constant velocity. The total energy of the system shows a non-linear increase with t once the deformation starts, which is expected from the typical dependency of bond energy with bond distance $U_b = k_b (r - r_0)^2$ (Fig. 9). As always, it is important to ensure that the simulation behaves as expected by opening the *dump.lammpstrj* file with VMD (Fig. 10).

Figure 10. CNT before (top) and after (bottom) deformation.

To import the YAML log file into Python and plot the total energy as a function of the number of step, create a new file within *unbreakable-bonds/* called *read-log.py*. Copy the following lines into it:

```
import numpy as np
import re, yaml

docs = ""
with open("log.lammps") as f:
    for line in f:
        pattern = (
            r"^(keywords:.*$|"
            r"data:$|"
            r"---$|"
            r"\.\.\.$|"
            r"  - \[.*\]$)"
        )
        m = re.search(pattern, line)
        if m:
            docs += m.group(0) + '\n'
thermo = list(yaml.load_all(docs,
                Loader=yaml.CSafeLoader))

Etot_vs_step = []
for line in thermo[0]['data']:
    Etot_vs_step.append([line[0], line[4]])
for line in thermo[1]['data']:
    Etot_vs_step.append([line[0], line[4]])
np.savetxt("etot_vs_step.dat", Etot_vs_step)
```

The *log* file is first open, a certain pattern is identified and stored as a string character named *docs*. The string is then converted in a list, and the number of steps as well as the total energy E_{tot} are extracted into the list named *Etot_vs_step*, which is saved in the data file named *etot_vs_step.dat* using the NumPy *savetxt* function. *thermo[0]* can be used to access the informations from the first minimization run, and *thermo[1]* from the second MD run.

3.2.2 Breakable bonds

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

Input file initialization

Create a second folder called *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
```

The 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*, since no explicit bond information is required with AIREBO.

Adapt the topology file

Since *bond*, *angle*, and *dihedral* do not need to be explicitly set when using AIREBO, some small changes need to be made to the previously generated *.data* file. Duplicate the previous file *cnt_molecular.data*, name the copy *cnt_atom.data*, and 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 placed 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 this:

```
(...)
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
```

Use of AIREBO potential

Then, let us import the LAMMPS data file, and set the pair coefficients by adding the following lines into *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 letter *C*. Download the *CH.airebo* file and place it 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 Ångstroms was used for the box size along the *z* axis, to allow for larger deformation. In addition, the *change_box* command was placed before the *displace_atoms* to avoid having the CNT crossing the edge of the box.

Start the simulation

Here, let us impose a constant velocity deformation using the atoms of one edge while maintaining the other edge fixed (note that for the unbreakable CNT, the motion was imposed on the 2 edges). As an equilibration step, let us set the velocity to 0 for the atoms of both edges. Add the following lines into the *input.lammps* file:

```
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 &
    cnt_length.dat
fix at2 all ave/time 10 10 100 f_mysf1[1] &
    f_mysf2[1] file edge_force.dat

dump mydmp all atom 1000 dump.lammpstrj

thermo 100
thermo_modify temp Tmid
```

Figure 11. CNT with broken bonds. Here, the *DynamicBonds* representation of VMD is used to properly visualize the bond breaking.

Figure 12. Evolution of the total energy E_{tot} of the CNT with time t. Here, the potential is AIREBO (i.e. the bonds are breakable).

```
timestep 0.0005
run 5000
```

Note the relatively small timestep of 0.0005 ps used. A reactive force field usually requires a smaller timestep than a 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.

Launch the deformation

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

```
velocity carbon_top set 0 0 0.15
run 280000
```

By opening the *lammpstrj* file using VMD, it is possible to observe the bonds breaking at approximately two-thirds of the simulation (Fig. 11). If the bonds do not break, use a longer run.

Looking at the evolution of energy again, one can see that the energy is increasing with the deformation. When bonds break, the energy relaxes abruptly, as can be seen near t = 110 ps and again near t = 130 ps in Fig. 12.

3.3 Tutorial 3: Polymer in water

The goal of this tutorial is to use LAMMPS to solvate a small hydrophilic polymer (PEG - PolyEthylene Glycol) in a reservoir of water (Fig. 13). Once the water reservoir is properly equilibrated at the desired temperature and pressure, the polymer molecule is added and a constant stretching force is applied to both ends of the polymer. The evolution of the

Figure 13. The polymer molecule (PEG - PolyEthylene Glycol) solvated in water as simulated during Tutorial 3. Water molecules are represented as a transparent continuum field for clarity.

polymer length is measured as a function of time. The GROMOS 54A7 force field [19] is used for the PEG, the SPC/Fw model [20] is used for the water, and the long-range Coulomb interactions are solved using the PPPM solver [21]. This tutorial was inspired by a publication by Liese and coworkers, in which molecular dynamics simulations are compared with force spectroscopy experiments [22].

3.3.1 Preparing the water reservoir

In this tutorial, the water reservoir is first prepared in the absence of the polymer. A rectangular box of water is created and equilibrated at ambient temperature and ambient pressure. The SPC/Fw water model is used [20], which is a flexible variant of the rigid SPC (simple point charge) model [23]. Create a folder called *pureH2O/*. Inside this folder, create an empty text file called *input.lammps*. Copy the following lines into it:

```
units real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style harmonic
pair_style lj/cut/coul/long 10
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, and 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/or 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*. Finally, the *special_bonds* command, which was already seen in tutorial 2, sets the LJ and Coulomb weighting factors for the interaction between neighboring atoms. With the *pair_style* named *lj/cut/coul/long*, atoms interact through both a Lennard-Jones (LJ) potential and Coulomb interactions. The value of 10 Å is the cutoff. Finally, the *kspace* command defines the long-range solver for the Coulomb interactions. The *pppm* style refers to particle-particle particle-mesh [21].

Then, let us create a 3D simulation box of dimensions $9 \times 3 \times 3$ nm^3, and make space for 9 atom types (2 for the water, 7 for the polymer), 7 bond types (1 for the water, 6 for the polymer), 8 angle types (1 for the water, 7 for the polymer), and 4 dihedral types (only for the polymer). Copy the following lines into *input.lammps*:

```
region box block -45 45 -15 15 -15 15
create_box 9 box &
bond/types 7 &
angle/types 8 &
dihedral/types 4 &
extra/bond/per/atom 3 &
extra/angle/per/atom 6 &
extra/dihedral/per/atom 10 &
extra/special/per/atom 14
```

The *extra/x/per/atom* commands are here for memory allocation. 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 *PARM.lammps* file next to the *pureH2O/* folder. Within *PARM.lammps*, the *mass* and *pair_coeff* of atoms of types 8 and 9 are for water and the atoms of types 1 to 7 are for the polymer molecule. Similarly, the *bond_coeff 7* and *angle_coeff 8* are for water, while all the other parameters are for the polymer.

Let us create water molecules. To do so, let us import a molecule template called *H2O-SPCFw.mol* and then randomly create 1050 molecules. Add the following lines into *input.lammps*:

```
molecule h2omol H2O-SPCFw.mol
create_atoms 0 random 1050 87910 NULL mol &
    h2omol 454756 overlap 1.0 maxtry 50
```

The *overlap 1* option of the *create_atoms* command ensures that no atoms are placed exactly in 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 cases, depending on the system and 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 3150 atoms
```

When LAMMPS fails to create the desired number of molecules, a WARNING appears in the *log* file. The molecule template called *H2O-SPCFw.mol* must 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, or the IDs of the atoms that are connected by bonds, angles, etc.

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

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

In general, resetting the step of the simulation to 0 using the *reset_timestep* command is optional. It is used here because the number of iterations performed by the *minimize* command is usually not a round number (since the minimization stops when one of four criteria is reached). Let us use the *fix npt* to control the temperature of the molecules with a Nosé-Hoover thermostat and the pressure of the system with a Nosé-Hoover barostat [24–26], by adding the following line into *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
```

Figure 14. Water reservoir after equilibration. Oxygen atoms are in red, and hydrogen atoms are in white.

Figure 15. Evolution of the density ρ of water with time t. The density ρ reaches a plateau after $t \approx 10$ ps.

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

Finally, let us set the timestep to 1.0 fs, and run the simulation for 20 ps by adding the following lines into *input.lammps*:

```
timestep 1.0
run 20000

write_data H2O.data
```

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 (see a snapshot of the equilibrated system in Fig. 14). You can also open the *density.dat* file to ensure that the system converged toward the density of liquid water during the 20 ps of simulation (Fig. 15).

3.3.2 Solvating the PEG in water

Now that the water reservoir is equilibrated, we can safely include the PEG polymer in the water. The PEG molecule topology was downloaded from the ATB repository [27, 28]. It has a formula $C_{28}H_{58}O_{15}$, and the parameters are taken from the GROMOS 54A7 force field [19].

Figure 16. The PEG molecule in vacuum. The carbon atoms are in gray, the oxygen atoms in red, and the hydrogen atoms in white.

Create a second folder alongside *pureH2O/* and call it *mergePEGH2O/*. Create a new blank file in it, call it *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 10
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 previously generated data file *H2O.data* as well as the *PARM.lammps* file:

```
read_data ../pureH2O/H2O.data &
    extra/bond/per/atom 3 &
    extra/angle/per/atom 6 &
    extra/dihedral/per/atom 10 &
    extra/special/per/atom 14
include ../PARM.lammps
```

Download the template called *PEG-GROMOS.mol* for the PEG molecule, and then create a single molecule in the middle of the box:

```
molecule pegmol PEG-GROMOS.mol
create_atoms 0 single 0 0 0 mol pegmol 454756
```

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

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

Here the value of 2 Ångstroms 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 is 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 rescaled 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 as well as the system temperature and volume:

Figure 17. A single PEG molecule in water. Water molecules are represented as a transparent continuum field for clarity.

```
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 30000
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 dimensions slightly evolve along *x* to accommodate the new configuration. In addition, the temperature remains close to the target value of 300 K throughout the entire simulation, and the enthalpy is almost constant, suggesting that the system was close to equilibrium from the start.

3.3.3 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 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. Copy the following line into the *input.lammps* file:

```
variable f0 equal 5
```

The force magnitude of 1 kcal/mol/Å corresponds to 67.2 pN and was 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). Then, copy the same lines as previously:

```
units real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style harmonic
pair_style lj/cut/coul/long 10
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-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
```

Add the following *dump* command to the input to print the atom positions every 1000 steps:

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

Let us use a single Nosé-Hoover thermostat applied to all the 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 and the gyration radius of the molecule [29] by adding the following lines to *input.lammps*:

```
variable mytemp equal temp
fix myat1 all ave/time 10 10 100 &
    v_mytemp file 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 end-to-end-distance.dat
compute rgyr PEG gyration
fix myat3 all ave/time 10 10 100 c_rgyr &
    file gyration-radius.dat
thermo 1000
```

Figure 18. PEG molecule stretched along the *x* direction in water. Water molecules are represented as a transparent continuum field for clarity.

Finally, let us simulate 30 picoseconds without any external forcing:

```
run 30000
```

This first run will serve as a benchmark to quantify later 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 topull1 addforce ${f0} 0 0
fix myaf2 topull2 addforce -${f0} 0 0
run 30000
```

Run the *input.lammps* file using LAMMPS. If you open the *dump.lammpstrj* file using *VMD*, you should see that the PEG molecule eventually aligns in the direction of the force (as seen in Fig. 18). The evolutions of the end-to-end distance and of the gyration radius over time show that the PEG is adjusting to the external forcing in less than 10 ps (Fig. 19).

3.4 Tutorial 4: Nanosheared electrolyte

The objective of this tutorial is to simulate an electrolyte nanoconfined and sheared by two walls (Fig. 20). The density and velocity profiles of the fluid in the direction normal to the walls are extracted to highlight the effect of confining a fluid on its local properties. This tutorial illustrates some key aspects of combining a fluid and a solid in the same simulation. A major difference from the previous tutorial, Polymer in water, is that here a rigid four-point water model named TIP4P is used [30]. TIP4P is one of the most common water models due to its high accuracy.

Figure 19. a) Evolution of the end-to-end distance $d_{\text{end-to-end}}$ of the PEG with time t. The forcing starts at t = 5 ps. b) Evolution of the gyration radius R_{gyr} of the PEG molecule.

Figure 20. The electrolyte confined in a nanometer slit pore as simulated during Tutorial 4. Na^+ ions are represented as purple spheres, Cl^- ions as cyan spheres, water molecules are colored in red and white, and the walls are colored in gray. The arrows indicate the imposed lateral motion of the walls.

3.4.1 System preparation

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

System generation

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

```
boundary p p f
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
kspace_modify slab 3.0
```

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 those in the previous tutorial, Polymer in water, with two major differences: the use of *lj/cut/tip4p/long* instead of *lj/cut/coul/long*, and *pppm/tip4p* instead of *pppm*. When using *lj/cut/tip4p/long* and *pppm/tip4p*, the interactions resemble the conventional Lennard-Jones and Coulomb interactions, except that they are specifically designed for the four-point water model. Therefore, LAMMPS automatically creates a four-point water molecule using atoms of type 1 as oxygen (*O*) and atoms of type 2 as hydrogen (*H*). The fourth massless atom (*M*) of the TIP4P water molecule does not have to be defined explicitly, and the value of 0.1546 Å corresponds to the *O-M* distance of the TIP4P-2005 water model [30]. All the other atoms in the simulation are treated normally with long-range Coulomb interactions. Another novelty, here, is the use of *kspace_modify slab 3.0* that is combined with the non-periodic boundaries along the *z* coordinate: *boundary p p f*. With the *slab* option, the system is treated as periodical along *z*, but with an empty volume inserted between the periodic images of the slab, and the interactions along *z* effectively turned off.

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

```
lattice fcc 4.04
region box block -3 3 -3 3 -5 5
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 a 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 in the positions of the previously defined lattice, thus forming fcc solids. To add the water molecules, first download the molecule template called *RigidH2O.txt* and place it within *systemcreation/*. The template contains all the necessary information concerning the water molecule, such as atom positions, bonds, and angles. 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
```

Within the last three lines, a *region* named *rliquid* is created based on the last defined lattice, *fcc 4.04*. *rliquid* will be used for depositing the water molecules. The *molecule* command opens up the molecule template called *RigidH2O.txt*, and names the associated molecule *h2omol*. The new molecules are placed 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 called *PARM.lammps* 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
pair_coeff 2 2 0.0 1.0
pair_coeff 3 3 0.04690 2.4299
pair_coeff 4 4 0.1500 4.04470
pair_coeff 5 5 11.697 2.574
pair_coeff 1 5 0.4 2.86645
```

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.

As already seen in previous tutorials and with the important exception of *pair_coeff 1 5*, only pairwise interactions between atoms of identical types was assigned. By default, LAMMPS calculates the pair coefficients for the interactions between atoms of different types (i and j) by using geometrical average: $\epsilon_{ij} = (\epsilon_{ii} + \epsilon_{jj})/2$, $\sigma_{ij} = (\sigma_{ii} + \sigma_{jj})/2$. If the default value of 5.941 kcal/mol was kept for $\epsilon_{1\text{-}5}$, the solid walls would be extremely hydrophilic, causing the water molecule to form dense layers. As a comparison, the water-water

energy $\epsilon_{1\text{-}1}$ is only 0.185199 kcal/mol. Therefore, the walls were made less hydrophilic by reducing the value of $\epsilon_{1\text{-}5}$. Copy the following lines into *PARM.lammps* as well:

```
bond_coeff 1 0 0.9572
angle_coeff 1 0 104.52
```

The *bond_coeff* command, used here for the O-H bond of the water molecule, sets both the spring constant of the harmonic potential and the equilibrium distance of 0.9572 Å. The constant can be 0 for a rigid water molecule because the shape of the molecule will be preserved by the SHAKE algorithm (see below) [31, 32]. Similarly, the angle coefficient for the H-O-H angle of the water molecule sets the force constant of the angular harmonic potential to 0 and the equilibrium angle to 104.52°.

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
```

As it is now, the fluid density within the two walls is too high. To avoid high density and pressure, let us add the following lines into *input.lammps* to delete to delete about 15 % of the water molecules:

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

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

```
run 0

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

With *run 0*, the simulation will run for 0 steps, which is enough for creating the system and saving the final state. The *write_data* creates a file called *system.data* containing the information required to restart the simulation from the final configuration generated by this input file. With the *nocoeff* option, the parameters from the force field are not written in the *.data* file. The *write_dump* command prints the final positions of the atoms into *dump.lammpstrj*, a file that can be opened with VMD to visualize the system. Run the *input.lammps* file using LAMMPS. Check that your system has been correctly created by also looking at the periodic images, just like in Fig. 21.

Figure 21. Side view of the system. Periodic images are represented in darker colors. 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.

Energy minimization

Let us move the atoms and place them in more energetically favorable positions before starting the actual molecular dynamics simulation. Let us call this step *energy minimization*, although it is not a conventional *minimization* as done for instance in the first tutorial; Lennard-Jones fluid. Instead, a molecular dynamics simulation will be performed here, with some techniques employed to prevent the system from exploding due to overlapping atoms.

Let us create a new folder called *minimization/* next to *systemcreation/*, and create a new input file called *input.lammps* in it. Copy the following lines in *input.lammps*:

```
boundary p p f
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
kspace_modify slab 3.0

read_data ../systemcreation/system.data

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

The only difference from 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) and 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* command performs constant NVE integration to update the positions and velocities of the atoms at each timestep. The difference is that *fix nve/limit* also limits the maximum distance atoms can travel at each timestep. Here, the chosen maximum distance is 0.1 Å. Because the *fix nve/limit* is applied to the group *fluid*, only the water 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 and control the printing of thermodynamic outputs by adding the following lines to *input.lammps*: ormal

```
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 higher 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 nocoeff
```

For the last of the three 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 (Fig. 22). If you look at the trajectory using VMD, you will see that some of the atoms are moving, particularly those that were initially in problematic positions.

Figure 22. Total energy of the system E_{tot} as a function of time t extracted from the log file using *Python* and *lammps_logfile* [33]. The vertical dashed lines demarcate the three consecutive steps.

System equilibration

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 f
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
kspace_modify slab 3.0

read_data ../minimization/system.data

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

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* does not influence the dynamics but will keep the system in the center of the box, which makes the visualization easier. Then, add the following lines into *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

Figure 23. Distance between the walls *d* as a function of time *t*.

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 nocoeff
```

Run the *input.lammps* file using LAMMPS. As seen from the data printed by *fix myat1*, the distance between the two walls reduces until it reaches an equilibrium value (Fig. 23).

Note that it is generally 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 (≈ 1.2 nm), i.e. on the order of half a nanosecond.

3.4.2 Imposed shearing

From the equilibrated configuration, let us impose a lateral motion on 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 f
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
kspace_modify slab 3.0
```

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 with the previous input is that, here, two thermostats are used, one for the fluid (*myber1*) and one for the solid (*myber2*). The use of *fix_modify* together with *compute temp* 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 thermalization, 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 commands 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 the 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 into *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] &
```

Figure 24. Velocity profiles for water molecules and ions (blue disks), and walls (orange squares) along the z axis. The line is a linear fit assuming that the pore size is h = 1.8 nm.

Figure 25. Water density ρ along the z axis.

```
    f_mysf2[1] file forces.dat

timestep 1.0
run 200000
write_data system.data nocoeff
```

Here, a binning of 1 Å is used for the density profiles generated by the *ave/chunk* commands. For smoother profiles, you can reduce the value of the bins. The averaged velocity and density profiles of the fluid are plotted in Figs.24-25. As expected for such Couette flow geometry, the velocity of the fluid is found to increase linearly along z.

From the force applied by the fluid on the solid, one can extract the stress within the fluid, which allows for the measurement of its viscosity $\dot{\eta}$ according to $\eta = \tau/\dot{\gamma}$ where τ is the stress applied by the fluid on the shearing wall, and $\dot{\gamma}$ the shear rate (which is imposed here) [34]. Here, the shear rate is approximatively $\dot{\gamma} = 16 \cdot 10^9\,\mathrm{s}^{-1}$, and using a surface area of $A = 6 \cdot 10^{-18}\,\mathrm{m}^2$, one gets an estimate for the shear viscosity for the confined fluid of η = 6.6 mPa.s. The viscosity calculated at such a high shear rate may differ from the expected *bulk* value. In general, it is recommended to use a lower value for the shear rate. Note that for lower shear rates, the ratio of noise-to-signal is larger, and longer simulations are needed. Another important point to keep in mind is that the viscosity of a fluid next to a solid surface is typically larger than in bulk due to interaction with the walls [35]. Therefore, one expects the present simulation to return a viscosity that is slightly larger than what would be measured in the absence of a wall.

Figure 26. A portion of the silicon dioxide structure as simulated during Tutorial 5. Atoms are colored by their charges.

3.5 Tutorial 5: Reactive silicon dioxide

The objective of this tutorial is to use the reactive force field ReaxFF to calculate the partial charges of a system undergoing deformation, as well as the formation and breaking of chemical bonds [36, 37]. The system simulated here is a block of silicon dioxide SiO_2 (Fig. 26) that is deformed until rupture. Particular attention is given to the evolution of the atomic charges during the deformation of the structure, and the chemical reactions occurring due to the deformation are tracked over time.

3.5.1 Prepare and relax

Create a folder, name it *RelaxSilica/*, and download the initial topology of a small amorphous silica structure called *silica.data*. This structure was created using a classical force field called Vashishta [38]. If you open the *silica.data* file, you will see in the Atoms section that all silicon atoms have the same charge q = 1.1 e, and all oxygen atoms have the charge q = –0.55 e. This is common with classical force fields and will change once ReaxFF is used.

The first action 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 parameters over time. Create an input file called *input.lammps* in *RelaxSilica/*, and copy the following lines into 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 copy three crucial lines into 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 *pair_style reaxff* is used with no control file. The *safezone* and *mincap* keywords have been added to avoid memory allocation issues, which sometimes can trigger the segmentation faults and *bondchk* failed errors. The *pair_coeff* uses the *reaxCHOFe.ff* file, which must be downloaded and saved within *RelaxSilica/*. For consistency with the masses and the *silica.data* file, 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 [39]. The charge equilibration occurs at every step. The values 0.0 and 10.0 are the low and the high cutoffs, respectively, and 1.0e–6 is a tolerance. Finally, *maxiter* sets an upper limit to the number of attempts to equilibrate the charge.

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 qSi equal charge(grpSi)/count(grpSi)
variable qO equal charge(grpO)/count(grpO)
```

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 into the *input.lammps*:

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

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 and some bonds are broken:

```
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 called *species.log*. Let us perform a very short run using the 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
```

Figure 27. Average charge per atom of the silicon q_{Si} (a) and oxygen q_O (b) atoms during equilibration, as given by the *v_qSi* and *v_qO* variables.

Figure 28. Volume of the system V as a function of time t.

```
timestep 0.5

run 5000

write_data silica-relaxed.data
```

Run the *input.lammps* file using LAMMPS. As seen from *species.log*, only one species is detected, called *Si192O384*, representing 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 environment in real time (Fig. 27). Additionally, the average charge of the atoms is strongly fluctuating at the beginning of the simulation. This early fluctuation correlates with a rapid volume change of the box (Fig. 28), during which the inter-atomic distances are expected to quickly change.

We can also plot the charge distribution $P(q)$, using the charge values printed in the *.lammptrj* file (Fig. 29). 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

Figure 29. Probability distribution of charge of silicon (positive, blue) and oxygen (negative, orange) atoms during equilibration.

Figure 30. A slice of the amorphous silica, where atoms are colored by their charges. Dangling oxygen groups appear in greenish, bulk Si atoms with a charge of about 1.8 e appear in red/orange, and bulk O atoms with a charge of about –0.9 e appear in blue. To color the atoms by their charge using VMD, use *Charge* as the coloring method in the representation windows, and then tune the *Color scale* in the *Color control windows*.

of the dangling oxygen groups) (Fig. 30).

3.5.2 Deform the structure

Let us apply a deformation to the structure to force some Si – O bonds to break (and eventually re-assemble). Next to *RelaxSilica/*, create a folder, call it *Deform/* and create a file called *input.lammps* in it. Copy the same lines as previously in *input.lammps*:

```
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
```

Figure 31. Evolution of the average charge per atom of the silicon q_{Si} (a) and oxygen q_O (b) over time t. The vertical dashed lines mark the beginning of the deformation, and the horizontal dotted lines denote the initial values for the average charge.

```
group grpO type 2
variable qSi equal charge(grpSi)/count(grpSi)
variable qO equal charge(grpO)/count(grpO)

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

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:

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

Here, no barostat is used because the box volume will be imposed by the *fix deform*.

Let us run for 5000 steps without deformation, then apply the *fix deform* for elongating progressively the box along x 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
```

Run the *input.lammps* file using LAMMPS. During the deformation, the charge values progressively evolve until the structure eventually breaks down. After the structure breaks

Figure 32. Evolution of the temperature T of the silica system over time t. The material ruptures near t = 10 ps.

Figure 33. Amorphous silicon oxide after deformation. The atoms are colored by their charges. Dangling oxygen groups appear in greenish, bulk Si atoms with a charge of about 1.8 e appear in red/orange, and bulk O atoms with a charge of about –0.9 e appear in blue.

down, the charges equilibrate near new average values that differ from the starting averages (Fig. 31). The difference between the initial and the final charges can be explained by the presence of defects as well as new solid/vacuum interfaces, and the fact that surface atoms typically have different charges compared to bulk atoms. There is also a strong increase in temperature during the rupture of the material (Fig. 32). At the end of the deformation, one can visualize the broken material using VMD. Notice the different charge values of the atoms located near the vacuum interfaces, compared to the atoms located in the bulk of the material (Fig. 33).

One can have a look at the charge distribution after deformation, as well as during the deformation (Fig. 34). As expected, the final charge distribution slightly differs from the previously calculated. In my case, no new species were formed during the simulation, as can be seen from the *species.log* file:

```
#  Timestep   No_Moles   No_Specs  Si1O2O384
       5          1          1          1
(...)
#  Timestep   No_Moles   No_Specs  Si1O2O384
   30000          1          1          1
```

Sometimes, O_2 molecules are formed during the deformation. If this is the case, a new column *O2* appears in the

Figure 34. Probability distribution of charge of silicon (positive, blue) and oxygen (negative, orange) after deformation. The stars correspond to the charge distribution during deformation.

species.log file.

3.5.3 Decorate the surface

In ambient conditions, some of the surface SiO_2 atoms are chemically passivated by forming covalent bonds with hydrogen (H) atom [40]. Let us add hydrogen atoms randomly to the cracked silica and observe how the system evolves. Next to *RelaxSilica/* and *Deform/*, create a folder, and call it *Decorate/*. Then, let us modify the previously generated data file *silica-deformed.data* and make space for a third atom type. Copy *silica-deformed.data* from the *Deform/* folder, and modify the first lines as follow:

```
576 atoms
3 atom types

-12.15958814509652 32.74516585669389 xlo xhi
2.316358282925984 18.26921942866687 ylo yhi
1.3959542953413138 19.189623416252907 zlo zhi

Masses

1 28.0855
2 15.999
3 1.008

(...)
```

Create a file called *input.lammps* into the *Decorate/* folder, and copy the following lines into it:

```
units real
atom_style full

read_data silica-deformed.data
displace_atoms all move -12 0 0 # optional

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

Here, the *displace_atoms* command was used to move the center of the crack near the center of the box. This step is optional but makes the visualization of the interface in VMD easier. A different value for the shift may be needed in your case, depending on the location of the crack. A difference with the previous input is that three atom types are specified in the *pair_coeff* command, *Si O H*, instead of two.

Then, let us adapt some familiar commands to measure the charges of all three types of atoms, and output the charge values into log files:

```
group grpSi type 1
group grpO type 2
group grpH type 3
variable qSi equal charge(grpSi)/count(grpSi)
variable qO equal charge(grpO)/count(grpO)
variable qH equal charge(grpH)/(count(grpH)+1e-10)

thermo 5
thermo_style custom step temp etotal press vol &
    v_qSi v_qO v_qH
fix myspec all reaxff/species 5 1 5 species.log &
    element Si O H
```

Here, the +1e – 10 was added to the denominator of the *variable qH* to avoid dividing by 0 at the beginning of the simulation. Finally, let us create a loop with 10 steps, and create two hydrogen atoms at random locations at every step:

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

label loop
variable a loop 10

variable seed equal 35672+${a}
create_atoms 3 random 2 ${seed} NULL &
    overlap 2.6 maxtry 50
group grpH type 3
run 2000
write_dump all custom dump.${a}.lammpstrj &
    id type q x y z

next a
jump SELF loop

write_data decorated.data
```

Here, a different *lammpstrj* file is created for each step of the loop to avoid creating dump files with varying numbers of atoms, which VMD can't read. Once the simulation is over, it can be seen from the *species.log* file that all the created hydrogen atoms reacted with the SiO_2 structure to form surface groups (such as hydroxyl (-OH) groups).

Figure 35. Cracked silicon oxide after the addition of hydrogen atoms. The atoms are colored by their charges. Dangling oxygen groups appear in greenish, bulk Si atoms with a charge of about 1.8 e appear in red/orange, and bulk O atoms with a charge of about –0.9 e appear in blue.

Figure 36. Water molecules adsorbed in cracked silica (SiO_2) material as simulated during Tutorial 6. Water molecules are colored in cyan and white, oxygen (O) atoms from SiO_2 in red, and silicon (Si) atoms in yellow.

```
# Timestep    No_Moles    No_Specs   Si192O384   H
  5           3           2          1           2
(...)
# Timestep    No_Moles  No_Specs  Si192O384H20
  20000       1         1         1
```

At the end of the simulation, hydroxyl groups can be seen at the interfaces (Fig. 35).

3.6 Tutorial 6: Water adsorption in silica

The objective of this tutorial is to combine molecular dynamics and grand canonical Monte Carlo simulations to compute the adsorption of water molecules in cracked silica material (Fig. 36). This tutorial illustrates the use of the grand canonical ensemble in molecular simulation, an open ensemble in which the number of atoms or molecules within the simulation box is not constant. When using the grand canonical ensemble, it is possible to impose the chemical potential (or pressure) of a given fluid in a nanoporous structure.

3.6.1 Generation of the silica block

Let us first generate a block of amorphous silica (SiO_2). 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/*. The initial data file for the SiO atoms called *SiO.data* must be downloaded and saved in *SilicaBlock/*. This data file contains the coordinates of the 9 atoms, their masses, and their charges.

Let us replicate these atoms using LAMMPS, and apply an annealing procedure to obtain a block of amorphous silica. Create a new input file called *input.lammps* in the *SilicaBlock/* folder, and copy the following lines into it:

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

The main difference from some of the previous tutorials is the use of the *Vashishta* pair style. The Vashishta potential is a bond-angle energy-based potential, it deduces the bonds between atoms from their relative positions [38].

Let us then import the system consisting of 9 atoms, and replicate it four times in all three directions of space, thus creating a system with 576 atoms. Add the following lines into *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 to print out the positions of the atoms every 5000 steps:

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

Download the SiO.1990.vashishta, and copy it within the *Potential/* folder.

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
```

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, and 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
```

Here, an isotropic barostat is used for the melted phase at T = 6000 K, and an anisotropic barostat is used for all following phases. With the anisotropic barostat, all three directions of space are adjusted independently from one another. An anisotropic barostat is usually a better choice for a solid phase. For a liquid or a gas, an 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 (which may be due to an initial bad configuration, probably harmless here), the temperature follows well the desired annealing procedure (Fig. 37). 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 (Fig. 38). After

Figure 37. Temperature T of the system during annealing. The vertical dashed lines mark the transition between the different phases of the simulation.

Figure 38. Box dimensions along x (blue), y (orange), and z (dark) during annealing. The vertical dashed lines mark the transition between the different phases of the simulation.

running the simulation, the final LAMMPS topology file called *amorphousSiO.data* will be located in *SilicaBlock/* (Fig. 39).

3.6.2 Cracking the silica

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

```
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
```

Figure 39. Final amorphous silica (SiO_2). The silicon atoms are represented in yellow, and the oxygen atoms in red.

Let us progressively increase the size of the box in the x direction, thus forcing the silica to deform and eventually crack. To do so, a loop based on the jump command is used. At every step of the loop, the box dimension over x will be multiplied by a scaling factor 1.005. Add the following lines into 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
```

The *fix nvt* is used to control the temperature of the system, while the *change_box* command imposes incremental deformations of the box. Different scaling factors or different numbers of steps can be used to generate different defects in the silica. After the expansion, a final equilibration step of a duration of 20 picoseconds is performed. If you look at the *dump.lammpstrj* file using VMD, you can see the expansion 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). The final system is shown in Fig. 40.

3.6.3 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 a given chemical potential μ, and multiple attempts to insert water molecules at random positions are made. Each attempt is either accepted or rejected based on energy considerations. Find more details in classical textbooks like Ref. 1.

Using hydrid potentials

Create a new folder called *Addingwater/*. Download and save the water molecule template called *H2O.mol* within

Figure 40. Block of silica after deformation. The silicon atoms are represented in yellow, and the oxygen atoms in red. Some holes are visible.

the *Addingwater/* folder. 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_2, and lj/cut/tip4p/long for TIP4P water model (here the TIP4P/2005 model is used [30]). Combining the two force fields is done using the *hybrid/overlay* pair style. 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 a rigid water model is used here.

Before going further, we also need to make a few changes 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 atom types in the simulations, with O and H of H_2O 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 *H2O.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 leads to failure. Note that here, most water molecules overlap with the silica. These overlapping water molecules will have to be deleted before starting the simulation (see below). 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
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 SiO_2 (type 2), and not to the O and H of H_2O, thanks to the NULL parameters used for atoms of types 3 and 4. Pair coefficients for the lj/cut/tip4p/long potential are defined between O(H_2O) and between H(H_2O) atoms, as well as between O(SiO_2)-O(H_2O) and Si(SiO_2)-O(H_2O). Therefore, the fluid-fluid and the fluid-solid interactions will be dealt with by the lj/cut/tip4p/long potential. 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.

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

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

GCMC simulation

To prepare for the GCMC simulation, let us make the first equilibration step by adding the following lines into *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
```

Two different thermostats are used for SiO_2 and H_2O, 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
```

The *tfac_insert* option ensures the correct estimate 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 μ = –0.5 eV, which corresponds roughly to ambient conditions, i.e. to a relative humidity RH $\approx$ 50 % [41].) The large pressure value

Figure 41. Number of water molecules N_{H2O} as a function of the time t. The dashed vertical line marks the beginning of the GCMC step.

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, the log file indicates that 840 atoms (i.e. 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 were immediately deleted, leaving 24 water molecules (again, 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. Once the crack is filled with water molecules, the number of molecules reaches a plateau (Figs. 41-42). The final number of molecules depends on the imposed pressure, temperature, and the interaction between water and silica (i.e. its hydrophilicity). 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.

3.7 Tutorial 7: Free energy calculation

The objective of this tutorial is to measure the free energy profile of particles across a barrier potential using two methods: free sampling and umbrella sampling [1, 2, 42] (Fig. 43). For simplicity and to reduce computation time, the barrier potential will be imposed on the atoms with an additional force, mimicking the presence of a repulsive area in the middle of the simulation box without the need to simulate extra atoms. The procedure is valid for more complex systems and can be adapted to many other situations, such as measuring the adsorption barrier near an interface, or for calculating a translocation barrier through a membrane [43–45].

Figure 42. Snapshot of the silica system after the adsorption of water molecules. The oxygen atoms of the water molecules are represented in cyan, the silicon atoms in yellow, and the oxygen atoms of the solid in red.

Figure 43. Atoms as simulated during Tutorial 7. Only the atom colored in pink feels the additional force used for the umbrella sampling method.

3.7.1 Method 1: Free sampling

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

$$\Delta G = -RT \ln(p/p_0), \tag{1}$$

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

Basic LAMMPS parameters

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

```
variable sigma equal 3.405
variable epsilon equal 0.238
variable U0 equal 1.5*${epsilon}
variable dlt equal 0.5
variable x0 equal 5.0

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: 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 [46]. It was calculated as $2^{1/6} \times 3.405$ where $3.405 = \sigma$. The potential is shifted to be equal to 0 at the cut-off using the *pair_modify*. The system of unit *real*, for which energy is in kcal/mol, distance in Ångstrom, or time in femtosecond, has been chosen for practical reasons: the WHAM algorithm used in the second part of the tutorial automatically assumes the energy to be in kcal/mol.

System creation and settings

Let us define the simulation box and randomly add atoms by copying the following lines into *input.lammps*:

```
region myreg block -25 25 -5 5 -25 25
create_box 1 myreg
create_atoms 1 random 60 341341 myreg &
    overlap 1.5 maxtry 50
```

Figure 44. Potential U given in Eq. (2) (a) and force F given in Eq. (3) (b) as a function of the coordinate x. Here, U_0 = 0.36 kcal/mol, δ = 0.5 Å, and x_0 = 5 Å.

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

The variables U_0, δ, and x_0 defined in the previous subsection are used here to create the repulsive potential, restricting the atoms from exploring the center of the box:

$$U = U_0 \left[\arctan\left(\frac{x+x_0}{\delta}\right) - \arctan\left(\frac{x-x_0}{\delta}\right)\right]. \tag{2}$$

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

$$F = \frac{U_0}{\delta}\left[\frac{1}{(x-x_0)^2/\delta^2+1} - \frac{1}{(x+x_0)^2/\delta^2+1}\right]. \tag{3}$$

Fig. 44 shows the potential U and force F along the x axis. With $U_0 = 1.5\epsilon = 0.36$ kcal/mol, U_0 is of the same order as the thermal energy $k_B T = 0.24$ kcal/mol, where $k_B = 0.002$ kcal/mol/K is the Boltzmann constant and $T = 119.8$ K (see below). In this case, particles are expected to regularly overcome the energy barrier thanks to the thermal agitation.

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, we will 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
```

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.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.

Data analysis

First, let us ensure that the initial equilibration of 1 ns is long enough by examining the *density_evolution.dat* file. Here, we can see that the number of atoms in the central region, n_{centrer}, evolves near its equilibrium value (which is close to 0) after about 0.1 ns (Fig. 45). See also a snapshot of the equilibrated system made using VMD in Fig. 46. One can also 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 (Fig. 47). Then, let us plot $-RT\ln(\rho/\rho_{\text{bulk}})$ (i.e. Eq. (1) where the pressure ratio p/p_{bulk} is replaced by the density ratio ρ/ρ_{bulk} assuming the system is an ideal gas) and compare it with the imposed potential U from Eq. (2) (Fig. 48). The value for the reference density ρ_{bulk} = 0.0033 Å^{-3} was estimated by measuring the density of the reservoir from the raw density profiles. The agreement between the MD results and the imposed energy profile is reasonable, despite some noise in the central part (where fewer data points are available due to the repulsive potential).

Figure 45. Evolution of the number of atoms n_{center} in the central region *mymes* as a function of time t during equilibration. The dark line serves as a guide for the eyes. Here, U_0 = 0.36 kcal/mol, δ = 0.5 Å, and x_0 = 5 Å.

Figure 46. Snapshot of the system. The density of atoms is lowest in the central part of the box, *mymes*, due to the additional force F. Here, U_0 = 0.36 kcal/mol, δ = 0.5 Å, and x_0 = 5 Å.

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

Figure 47. Fluid density ρ along the x direction. Here, U_0 = 0.36 kcal/mol, δ = 0.5 Å, x_0 = 5 Å, and reference density ρ_{bulk} = 0.0033 Å^{-3}.

Figure 48. Potential U as a function of x measured using free sampling (blue disks) compared to the imposed potential given in Eq. (2) (dark line). Here, U_0 = 0.36 kcal/mol, δ = 0.5 Å, and x_0 = 5 Å.

a good resolution for the free energy profile. For instance, when we run the same simulation using $U_0 = 10\epsilon$, which corresponds to a situation where $U_0 \approx 10 k_B T$, not a single atom explores the central part of the simulation box during the 8 ns of simulation. In this case, using an enhanced sampling method is preferred; see the next section.

3.7.2 Method 2: Umbrella sampling

Umbrella sampling is a biased molecular dynamics method in which additional forces are added to a chosen atom to force it to explore the more unfavorable areas of the system [1, 2, 42]. Here, to encourage one of the atoms to explore the central region of the box, we apply a potential V and force it to move along the x-axis. The chosen path is called the axis of reaction. Several simulations (called windows) will be conducted with varying positions for the center of the applied biasing. The results will be analyzed using the weighted histogram analysis method (WHAM) [47], which allows for the removal of the biasing effect and ultimately deduces the unbiased free energy profile.

LAMMPS input script

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

```
variable sigma equal 3.405
variable epsilon equal 0.238
variable U0 equal 10*${epsilon}
variable dlt equal 0.5
variable x0 equal 5.0
variable k equal 1.5

units real
atom_style atomic
pair_style lj/cut 3.822
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 59 random 5 341341 myreg &
    overlap 1.5 maxtry 50

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

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 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 that of Method 1, except for the additional particle of type 2. Particles of type 1 and 2 are identical, having the same mass and Lennard-Jones parameters. However, the particle of type 2 will additionally be exposed to the biasing potential V (Fig.)fig:US-system-biased).

Let us create a loop with 50 steps, and move progressively the center of the bias potential by an increment of 0.1 nm. Add the following lines to *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} &
    ${xdes} 0 0 0
run 200000
fix myat1 all ave/time 10 10 100 v_xave &
    v_xdes file data/position.${a}.dat
run 1000000
unfix myat1
next a
jump SELF loop
```

A folder called *data/* needs to be created within *BiasedSam-*

Figure 49. Snapshot of the system with atoms of type 1 in cyan and the single atom of type 2 in pink. Only the atom of type 2 explores the central part of the box, *mymes*, due to its additional biasing potential V. Here, U_0 = 2.38 kcal/mol, δ = 0.5 Å, and x_0 = 5 Å.

pling/. The spring command serves to impose the additional harmonic potential V with the spring constant k. Note that the value of k should be chosen with care, if k is too small, the particle won't follow the biasing potential center, and if k is too large, there will be no overlapping between the different windows. The center 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. Run the *input.lammps* file using LAMMPS.

WHAM algorithm

To generate the free energy profile from the density distribution, let us use the WHAM algorithm as implemented by Marc Grossfield [48]. You can download it from Alan Grossfield website, and compile it by typing in the terminal:

```
cd wham
make clean
make
```

The compilation creates an executable called *wham* that you can copy in the *BiasedSampling/* folder. 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.

Download the *metadata.dat* file I have written and save it in *BiasedSampling/*. 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 is the number of bins, 1e-8 is the tolerance, and 119.8 is the temperature. A file called *PMF.dat* has been created, containing the free energy profile in kcal/mol. Again, one can compare the result of the PMF with the imposed potential U, demonstrating excellent agreement (Fig. 50). We can see that the agreement

Figure 50. Potential U as a function of x measured using umbrella sampling (blue disks) compared to the imposed potential given in Eq. (2) (dark line). Here, U_0 = 2.38 kcal/mol, δ = 0.5 Å, and x_0 = 5 Å.

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 performing extremely long and costly simulations.

Author Contributions

SG conceived and wrote all the online tutorials and underlying Sphinx documentation for lammpstutorials.github.io.

Potentially Conflicting Interests

There are no conflicting interests to declare.

Funding Information

S.G. acknowledges funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement N° 101065060.

Author Information

ORCID:
Simon Gravelle: 0000-0003-2149-6706

References

[1] **Frenkel D**, Smit B. Understanding molecular simulation: from algorithms to applications. Elsevier; 2023.

[2] **Allen MP**, Tildesley DJ. Computer simulation of liquids. Oxford university press; 2017.

[3] **van Gunsteren WF**, Dolenc J, Mark AE. Molecular simulation as an aid to experimentalists. Current opinion in structural biology. 2008; 18(2):149–153.

[4] **Thompson AP**, Aktulga HM, Berger R, Bolintineanu DS, Brown WM, Crozier PS, in't Veld PJ, Kohlmeyer A, Moore SG, Nguyen TD, et al. LAMMPS-a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. Computer Physics Communications. 2022; 271:108171.

[5] **van Gunsteren WF**, Daura X, Hansen N, Mark AE, Oostenbrink C, Riniker S, Smith LJ. Validation of molecular simulation: an overview of issues. Angewandte Chemie International Edition. 2018; 57(4):884–902.

[6] **Barrat JL**, Hansen JP. Basic concepts for simple and complex liquids. Cambridge University Press; 2003.

[7] **Hansen JP**, McDonald IR. Theory of simple liquids: with applications to soft matter. Academic press; 2013.

[8] **Humphrey W**, Dalke A, Schulten K. VMD: visual molecular dynamics. Journal of molecular graphics. 1996; 14(1):33–38.

[9] **van Rossum G**, Drake Jr FL. Python reference manual. Centrum voor Wiskunde en Informatica Amsterdam; 1995.

[10] **Hunter JD**. Matplotlib: A 2D graphics environment. Computing in Science & Engineering. 2007; 9(3):90–95.

[11] **Schneider T**, Stoll E. Molecular-dynamics study of a three-dimensional one-component model for distortive phase transitions. Physical Review B. 1978; 17(3):1302.

[12] **Wang X**, Ramírez-Hinestrosa S, Dobnikar J, Frenkel D. The Lennard-Jones potential: when (not) to use it. Physical Chemistry Chemical Physics. 2020; 22(19):10624–10633.

[13] **Fischer J**, Wendland M. On the history of key empirical intermolecular potentials. Fluid Phase Equilibria. 2023; 573:113876.

[14] **Hestenes MR**, Stiefel E, et al. Methods of conjugate gradients for solving linear systems, vol. 49. NBS Washington, DC; 1952.

[15] **Stuart SJ**, Tutein AB, Harrison JA. A reactive potential for hydrocarbons with intermolecular interactions. The Journal of chemical physics. 2000; 112(14):6472–6486.

[16] **Kohlmeyer A**, Vermaas J, TopoTools: Release 1.9. Zenodo; 2021. https://doi.org/10.5281/zenodo.598373.

[17] **Jorgensen WL**, Maxwell DS, Tirado-Rives J. Development and Testing of the OPLS All-Atom Force Field on Conformational Energetics and Properties of Organic Liquids. Journal of the American Chemical Society. 1996; 118(45):11225–11236.

[18] **Berendsen HJ**, Postma J, van Gunsteren WF, DiNola A, Haak JR. Molecular dynamics with coupling to an external bath. The Journal of chemical physics. 1984; 81(8):3684–3690.

[19] **Schmid N**, Eichenberger AP, Choutko A, Riniker S, Winger M, Mark AE, van Gunsteren WF. Definition and testing of the GROMOS force-field versions 54A7 and 54B7. European biophysics journal. 2011; 40:843–856.

[20] **Wu Y**, Tepper HL, Voth GA. Flexible simple point-charge water model with improved liquid-state properties. The Journal of chemical physics. 2006; 124(2).

[21] **Luty BA**, van Gunsteren WF. Calculating electrostatic interactions using the particle-particle particle-mesh method with nonperiodic long-range interactions. The Journal of Physical Chemistry. 1996; 100(7):2581–2587.

[22] **Liese S**, Gensler M, Krysiak S, Schwarzl R, Achazi A, Paulus B, Hugel T, Rabe JP, Netz RR. Hydration effects turn a highly stretched polymer from an entropic into an energetic spring. ACS nano. 2017; 11(1):702–712.

[23] **Berendsen HJ**, Postma JP, van Gunsteren WF, Hermans J. Interaction models for water in relation to protein hydration. In: *Intermolecular forces: proceedings of the fourteenth Jerusalem symposium on quantum chemistry and biochemistry held in jerusalem, israel, april 13–16, 1981* Springer; 1981. p. 331–342.

[24] **Nosé S**. A unified formulation of the constant temperature molecular dynamics methods. The Journal of chemical physics. 1984; 81(1):511–519.

[25] **Hoover WG**. Canonical dynamics: Equilibrium phase-space distributions. Physical review A. 1985; 31(3):1695.

[26] **Martyna GJ**, Tobias DJ, Klein ML. Constant pressure molecular dynamics algorithms. The Journal of chemical physics. 1994; 101(5):4177–4189.

[27] **Malde AK**, Zuo L, Breeze M, Stroet M, Poger D, Nair PC, Oostenbrink C, Mark AE. An automated force field topology builder (ATB) and repository: version 1.0. Journal of chemical theory and computation. 2011; 7(12):4026–4037.

[28] **Oostenbrink C**, Villa A, Mark AE, van Gunsteren WF. A biomolecular force field based on the free enthalpy of hydration and solvation: the GROMOS force-field parameter sets 53A5 and 53A6. Journal of computational chemistry. 2004; 25(13):1656–1676.

[29] **Fixman M**. Radius of Gyration of Polymer Chains. The Journal of Chemical Physics. 1962; 36(2):306–310.

[30] **Abascal JL**, Vega C. A general purpose model for the condensed phases of water: TIP4P/2005. The Journal of chemical physics. 2005; 123(23).

[31] **Ryckaert JP**, Ciccotti G, Berendsen HJ. Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes. Journal of computational physics. 1977; 23(3):327–341.

[32] **Andersen HC**. Rattle: A "velocity" version of the shake algorithm for molecular dynamics calculations. Journal of computational Physics. 1983; 52(1):24–34.

[33] **Sveinsson HA**, LAMMPS logfile reader. GitHub; 2021. https://github.com/henriasv/lammps-logfile.

[34] **Gravelle S**, Kamal C, Botto L. Violations of Jeffery's theory in the dynamics of nanographene in shear flow. Physical Review Fluids. 2021; 6(3):034303.

[35] **Wolde-Kidan A**, Netz RR. Interplay of Interfacial Viscosity, Specific-Ion, and Impurity Adsorption Determines Zeta Potentials of Phospholipid Membranes. Langmuir. 2021; 37(28):8463–8473.

[36] **van Duin AC**, Dasgupta S, Lorant F, Goddard WA. ReaxFF: a reactive force field for hydrocarbons. The Journal of Physical Chemistry A. 2001; 105(41):9396–9409.

[37] **Zou C**, van Duin A. Investigation of complex iron surface catalytic chemistry using the ReaxFF reactive force field method. Jom. 2012; 64:1426–1437.

[38] **Vashishta P**, Kalia RK, Rino JP, Ebbsjö I. Interaction potential for SiO2: A molecular-dynamics study of structural correlations. Physical Review B. 1990; 41(17):12197.

[39] **Rappe AK**, Goddard III WA. Charge equilibration for molecular dynamics simulations. The Journal of Physical Chemistry. 1991; 95(8):3358–3363.

[40] **Sulpizi M**, Gaigeot MP, Sprik M. The silica–water interface: how the silanols determine the surface acidity and modulate the water properties. Journal of chemical theory and computation. 2012; 8(3):1037–1047.

[41] **Gravelle S**, Dumais J. A multi-scale model for fluid transport through a bio-inspired passive valve. The Journal of Chemical Physics. 2020; 152(1).

[42] **Kästner J**. Umbrella sampling. Wiley Interdisciplinary Reviews: Computational Molecular Science. 2011; 1(6):932–942.

[43] **Wilson MA**, Pohorille A. Adsorption and solvation of ethanol at the water liquid- vapor interface: a molecular dynamics study. The Journal of Physical Chemistry B. 1997; 101(16):3130–3135.

[44] **Makarov DE**. Computer simulations and theory of protein translocation. Accounts of chemical research. 2009; 42(2):281–289.

[45] **Gravelle S**, Botto L. Adsorption of single and multiple graphene-oxide nanoparticles at a water–vapor interface. Langmuir. 2021; 37(45):13322–13330.

[46] **Weeks JD**, Chandler D, Andersen HC. Role of repulsive forces in determining the equilibrium structure of simple liquids. The Journal of chemical physics. 1971; 54(12):5237–5247.

[47] **Kumar S**, Rosenberg JM, Bouzida D, Swendsen RH, Kollman PA. The weighted histogram analysis method for free-energy calculations on biomolecules. I. The method. Journal of computational chemistry. 1992; 13(8):1011–1021.

[48] **Grossfield A**, An implementation of WHAM: the Weighted Histogram Analysis Method Version 2.0.10;. membrane.urmc.rochester.edu/content/wham/.