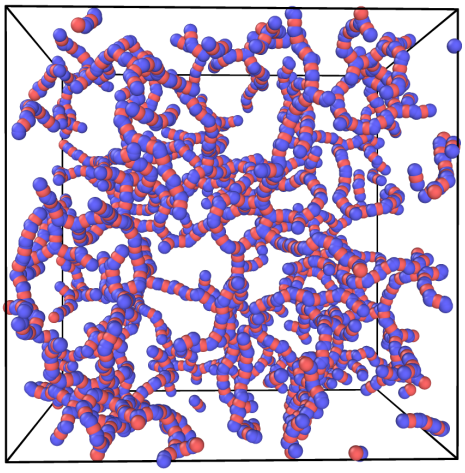
MTEK0033 Multiscale Modelling, University of Turku, January 2026

# Brownian Dynamics Simulation
# Exercises and Manual



Created by: Vikki Anand Varma
Supervised by: Prof. Alberto Scacchi

If you come across any problems, please email at vavarm@utu.fi

# Table of Contents


**1 A quick summary of the exercises** **2**

**2 Exercises** **3**

**3 Manual** **11**

3.1 Brownian Dynamics and it's characteristics . . . . . 11

3.1.1 System Setup . . . . . 11

3.1.2 Instructions . . . . . 13

3.1.3 Learning Outcome . . . . . 14

3.2 Studying the self-assembly in patchy-particle model . . . . . 15

3.2.1 System Setup . . . . . 15

3.2.2 Instructions . . . . . 21

3.2.3 Learning Outcomes . . . . . 21

3.3 Viscosity of a Lennard-Jones Fluid Using the Green-Kubo Relation . 22

3.3.1 System setup . . . . . 22

3.3.2 Instructions . . . . . 27

3.3.3 Learning Outcomes . . . . . 27

3.4 Calculation of liquid-vapor interface in Lennard-Jones (LJ) fluid . . 29

3.4.1 System Setup . . . . . 29

3.4.2 Instructions . . . . . 31

3.4.3 Learning Outcome . . . . . 32

3.5 Shear Viscosity of a Lennard-Jones Fluid Using Non-Equilibrium MD 33

3.5.1 Simulation setup . . . . . 33

3.5.2 Instructions . . . . . 38

3.5.3 Learning Outcome . . . . . 39

3.6 Langevin Dynamics and Fokker-Planck Equation . . . . . 40

3.6.1 System Setup . . . . . 40

3.6.2 Instructions . . . . . 41

3.6.3 Learning Outcome . . . . . 42

**4 References** **43**

# 1 A quick summary of the exercises

These exercises will give you a basic introduction to **Brownian Dynamics (BD) simulations**. The LAMMPS software will be used for the calculations, while OVITO will be used for visualisation of particle trajectories and assemblies.
The exercises focus on the **self-assembly of patchy particles**, examining how directional interactions and thermal fluctuations lead to chain formation, clusters, or gel-like structures. Additional exercises explore **viscosity under shear**, validation of **Einstein Stocks diffusivity relation** through simulation, relation between discrete and the continuum models etc.

## Brownian Dynamics (BD)

Brownian Dynamics is a stochastic simulation technique for modeling the **diffusive motion of particles in a solvent**, ignoring explicit solvent molecules. Particle motion is influenced by a combination of **conservative forces**, **drag**, and **random thermal forces** arising from the solvent.
In these exercises, each particle is modeled as a **patchy particle**: a central core decorated with attractive patches at specific positions on its surface. These anisotropic interactions induce **self-assembly** into chains, networks, or clusters depending on temperature, patch number, and interaction strength.
BD simulations solve the **overdamped Langevin equation**:

$$\gamma \frac{d\mathbf{r}_i}{dt} = -\nabla U_i + \mathbf{R}_i(t)$$

where $\gamma$ is the friction coefficient, $U_i$ is the potential acting on particle $i$, and $\mathbf{R}_i(t)$ is a random force with zero mean and variance given by the **Einstein relation**:

$$\langle R_{i\alpha}(t) R_{j\beta}(t') \rangle = 2k_B T \gamma \delta_{ij} \delta_{\alpha\beta} \delta(t - t')$$

This ensures proper thermal equilibrium and correct diffusive behavior.
In addition to self-assembly, the exercises explore **rheological properties** by applying shear and measuring the **viscous response** of the particle system. Students will learn to relate microscopic motion to macroscopic transport properties using Brownian Dynamics trajectories.
**Reference:** Non-equilibrium Statistical Mechanics by Robert Zwanzig.

**Note: If you are already familiar with LAMMPS commands, you may skip the system setup section that describes the individual commands. You can directly open the input files provided in the Instructions and Exercises sections and perform the calculations by following the procedure you learned to solve your problem.**

# Exercises

February 06, 2026

**Read the entire question paper carefully, before attempting to solve the problems to avoid any mistakes, and possible marks reduction. Answer the following questions.**

Maximum marks: 70

## 2 Exercises

### Utilities (General):

- Open the following link to learn how to run LAMMPS codes on CSC: Instructions for running LAMMPS on CSC.
- Open the following link to see the installation instructions for OVITO: OVITO installation instructions.

1. Verify the validity of the Einstein diffusion equation for Brownian motion, given by

$$D = \frac{k_B T}{\gamma},$$

where $D$ is the translational diffusion coefficient, $k_B$ is the Boltzmann constant, $T$ is the temperature, and $\gamma$ is the friction coefficient.

**Utilities**

- **Core task:** To compute diffusivity for different system configuration and see how the equation of motion produces correct theoretical results and hence verifying underlying algorithm of the simulation.
- **Task instructions (mandatory to check):** Manual Sec. 3.1.2
- **Relevant Folder:** `Exercises/Exercise_1`
- **LAMMPS Script details (if familiar with, can be skipped):** Manual Sec. 3.1

- **Step 1:**

  **Tasks:**

  (a) Edit and run the script:
  `Exercises/Exercise_1/gamma_vs_d/0.1, 0.3, 0.6, 1.0/brownian_dynamics.in`
  for the values,

  $$\gamma^{-1} = 0.1,\ 0.3,\ 0.6,\ 1.0,$$

  at a fixed mass $m = 1$ and $k_B T = 1.0$, while randomly changing any seed value used in the script to some integer, for each running script. It will ensure that each student has their own unique results.

  (b) Run the python script `msd_data_rectifier_and_plotter.py`, already provided in the relevant folder. Note down the diffusivity value shown on the terminal.

  (c) Edit manually the file named `Exercise_1/gamma_vs_d.txt` and save the value of diffusivity in the second column and in the row of the relevant $\gamma^{-1}$ value.

  (d) Paste content of the saved .txt file in your answer sheet.

  **[2.5 marks]**

- **Step 2:**

  **Tasks:**

  (a) Edit and run the script:
  `Exercises/Exercise_1/t_vs_d/1, 2, 3, 7/brownian_dynamics.in`
  for the values,

  $$k_B T = 1,\ 2,\ 3,\ 7,$$

  at a fixed mass $m = 1$ and $\gamma^{-1} = 1.0$, while randomly changing any seed value used in the script to some integer, for each running script. It will ensure that each student has their own unique results.

  (b) Run the python script `msd_data_rectifier_and_plotter.py`, already provided in the relevant folder. Note down the diffusivity value shown on the terminal.

  (c) Edit manually the file named `Exercise_1/t_vs_d.txt` and save the value of diffusivity in the second column and in the row of the relevant $k_B t$ value.

  (d) Paste content of the saved .txt file in your answer sheet.

  -

  **[2.5 marks]**

- **Step 3:**

  **Tasks:**

  (a) Edit and run the script:
  `Exercises/Exercise_1/m_vs_d/1, 2, 3, 6/brownian_dynamics.in`
  for the values,

  $$m = 1,\ 2,\ 3,\ 6,$$

at a fixed mass $k_B T = 1$ and $\gamma^{-1} = 1.0$ n, while randomly changing any seed value used in the script to some integer, for each running script. It will ensure that each student has their own unique results.

(b) Run the python script `msd_data_rectifier_and_plotter.py`, already provided in the relevant folder. Note down the diffusivity value shown on the terminal.

(c) Edit manually the file named `Exercise_1/m_vs_d.txt` and save the value of diffusivity in the second column and in the row of the relevant $m$ value.

(d) Paste the content of the saved .txt files in your answer sheet.

**[2.5 marks]**

- **Step 4:**

  **Tasks:**

  (a) Use the python script `Exercise_1/plotter.py` to export all the relevant plots, at once.

  (b) Paste your plots to the answer sheet for all the three previous observations.

  (c) Based on your plots, discuss whether the relations

  $$D \text{ vs. } \gamma^{-1}, \qquad D \text{ vs. } m^{-1}, \qquad D \text{ vs. } k_B T$$

  are linear. Clearly describe your observation.

**[2.5 marks]**

2. Simulate the patchy particle subunits with dumbbell shaped geometry and observe the state points e.g., temperature and volume fraction for the relvent phase.

**Utilities**

- **Core task:** To compute manually, the thermodynamic state point, e.g, temperature for a given phase and check its stability.
- **Task instructions (mandatory to check):** Manual Sec. 3.2.2
- **Relevant Folder:** `Exercises/Exercise_2`
- **LAMMPS Script details (if familiar with, can be skipped):** Manual Sec. 3.2

- **Step 1:**

  **Tasks**

  (a) Edit the molecule file `Exercises/Exercise_2/patchy_particle.mol`. Change the number of atoms, types and coordinates to make it dumbbell shaped such that at the center, the particle is supposed to be type 1 and at the poles particles are

supposed to be type 2. Specify the coordinates and types in the same order in an ascending manner. The location of the patches on the poles must be at a distance of 0.5 from the coordinate of the core atom.

(b) Paste your mole template file in answer sheet.

**[5 marks]**

- **Step 2:**

  **Tasks:**

  (a) Replace the old .mol file and paste the same newly constructed mole template file to all the folders given for different temperature configuration as: `Exercises/Exercise_2/temperature_vs_phase/0.1, 0.5, 1.0, 5.0`.

  (b) Edit the LAMMPS manuscript `patchy_particle.in`, already provided in all those relevant folders to the temperature values specified as $k_B T = 0.1, 0.5, 1.0, 5.0$.

  (c) Run the simulation and load the output data .lammpstrj to ovito to visulize.

  (d) Identify at least one system has chain like configuration and paste the screenshot of the same system in your answer sheet and report the corresponding temperature.

  **[5 marks]**

- **Step 3:**

  **Tasks:**

  (a) Observing your system, write a maximum 200 words description about the stability of the chain structure? Are they stable or not in compare to to other structures (e.g, globular structures obtained at $k_B T = 1.0$), specify some possible reasons?

  - **[5 marks]**

3. Modify the given LAMMPS input template to compute the viscosity of a Lennard-Jones fluid at a given density $\rho$.

   **Using the Green-Kubo relation. The shear viscosity is defined by**

$$\eta = \frac{V}{k_B T} \int_0^\infty \langle P_{\alpha\beta}(0)\, P_{\alpha\beta}(t) \rangle \, dt,$$

   where $P_{\alpha\beta}$ are the off-diagonal components of the pressure tensor, $V$ is the system volume, and $T$ is the temperature.

   **Utilities**

   - **Core task:** To LEARN the crucial debugging steps in LAMMPS and compute the viscosity using the above given Green Kubo Formula.
   - **Task instructions (mandatory to check):** Manual Sec. 3.3.2

- **Relevant Folder:** `Exercises/Exercise_3`
- **LAMMPS Script details (if familiar with, can be skipped):** Manual Sec. 3.3

- **Step 1:**

  **Tasks:**

  (a) Edit the script: `Exercises/Exercise_3/temperature_vs_viscosity/0.1/viscosity_input.in` such that the system generated number of particles satisfy the value $\rho = 0.1$.

  (b) Check the number of particles generate in the LAMMPS log output file (the given LAMMPS script is supposed to fail, however before failing the code, it will print the number of particle generated in the system).

  (c) Compute the density of your system using the specified box size and the created number of atoms.

  (d) Run the code and check if segmentation fault is happening.

  (e) Use trial and error method to change the equilibration run timestep to see, if code is running. Write down the reason for the failure of the code, after getting first successful run.

  (f) Paste the relevant section to your answer sheet with the value which leads to successful run.

  **[5 marks]**

- **Step 2:**

  (a) Edit the script: `Exercises/Exercise_3/temperature_vs_viscosity/0.5 viscosity_input.in` such that the system generated number of particles satisfy the value $\rho = 0.5$.

  (b) Check the number of particles generate in the LAMMPS log output file (the given LAMMPS script is supposed to fail, however before failing the code, it will print the number of particle generated in the system).

  (c) Compute the density of your system using the specified box size and the created number of atoms.

  (d) Use trial and error method and change the box size to see, if code is running without failing (box size in all the three direction should not be varied beyond the limit 20-25). Write down the reason for the failure of the code, after getting first successful run.

  (e) Paste the relevant section to your answer sheet with the value which leads to successful run.

  **Note:** 10%**, error in density is allowed, in case if parameters adjustment is not leading to an accurate particle number generation to get the target number density, for the given box size.**

  **[5 marks]**

- **Step 3:**

  **Tasks:**

  (a) Edit the script: `Exercises/Exercise_3/temperature_vs_viscosity/0.05, 0.1, 0.3,/viscosity_input.in` such that the system generated number of particles satisfy the value $\rho = 0.05, 0.1, 0.3$, respectively.

  (b) Run the LAMMPS script and check the log file which will show the value of viscosity at the end.

  (c) Edit the .txt file `Exercises/Exercise_3/viscosity_vs_density` and write down the density and the corresponding viscosity in the first and the second column respectively.

  (d) Run the python script `Exercises/Exercise_3/plotter.py` to plot the trend.

  (e) Paste the plot to your answer sheet and discuss about the trend.

  **[5 marks]**

- **Step 4:**

  The shear viscosity obtained from the Green–Kubo calculation in LAMMPS is reported in reduced Lennard–Jones (LJ) units.

  In Lennard–Jones (LJ) reduced units, viscosity is expressed in terms of the fundamental LJ units of mass $m$, energy $\epsilon$, and length $\sigma$. The dimensional form of viscosity is

  $$[\eta] = \frac{\text{mass}}{\text{length}\,\text{time}}.$$

  In LJ units, the characteristic time scale is defined as

  $$\tau = \sigma\sqrt{\frac{m}{\epsilon}}.$$

  and energy is defined as,

  $$\epsilon = k_{\mathrm{B}}T.$$

  Assume the following mappings between reduced LJ units and physical units: Length scale: $\sigma = 1\ \text{Å} = 1 \times 10^{-10}\,\text{m}$. Energy scale: $\epsilon = k_{\mathrm{B}}T$, with $T = 300\,\text{K}$, where the Boltzmann constant is $k_{\mathrm{B}} = 1.380649 \times 10^{-23}\,\text{J K}^{-1}$. Mass scale: corresponding to a single carbon atom, $m = 12$ amu, and 1 amu $= 1.660539 \times 10^{-27}\,\text{kg}$

  **Tasks:**

  (a) Derive the conversion factor that must be multiplied with the viscosity expressed in LJ units to obtain viscosity in SI units (Pa·s). **[2.5 marks]**

  (b) Clearly state the final numerical value of the conversion factor in SI units.

  - **[2.5 marks]**

4. Find the gas-liquid coexistence density at temperature $k_B T = 0.7$.

**Utilities**

- **Core task:** To compute manually, gas liquid coexistence points and have an experience with the interface study.
- **Task instructions (mandatory to check):** Manual Sec. 3.4.2
- **Relevant Folder:** `Exercises/Exercise_4`
- **LAMMPS Script details (if familiar with, can be skipped):** Manual Sec. 3.4

- **Step 1:**

  **Tasks:**

  (a) Edit all the input script with `.in` extension to set the asked temperature.

  (b) Run the script :
  `Exercises/Exercise_4/ liquid_box_creator.in` to generate the high-density liquid. Proper liquid density prevents bubble growth during gas-liquid simulation.

  (c) In the same folder run `intermediate.in` to remove periodicity along the $z$-axis from
  `compressed_coordinate.data`.

  (d) Open the data file and extend the $z$-axis symmetrically, e.g., change `38 52 zlo zhi` to `20 70 zlo zhi`. Save the file.

  (e) Run `coexistence_simulator.in` to equilibrate the system and record particle counts in bins along $z$. Output is stored in `bin_particles.LAMMPStrj`.

  (f) Run `data_rectifier.py` to average particle counts over time and produce the density profile $\rho(z)$ in `averaged_density_profile.txt`.

  (g) Open DESMOS or any online plotter and type $tanh(x)$ to observe the characteristic interface shape, similar to the gas-liquid interface.

  (h) Run `density_profile_fitter.py` to fit $\rho(z)$ to a $tanh$ function. The script outputs gas and liquid densities and a plot `density_profile_fit.png`.

  **[15 marks]**

- **Step 2:**

  **Tasks:**

  (a) Write down the coexistence density for both liquid and solid phases and produce the density profile plot and paste in the answer sheet.

  (b) What will happen at the higher temperature? Does the interface stability defined by surface tension will increase or decrease? Point out a few reasons.

  **[5 marks]**

5. **Suppose you are using the Martini force-field, which employs a coarse-grained carbon-carbon interaction potential, where each particle represents four carbon atoms. Now, answer the following questions?**

(a) Determine the appropriate bond length in a chain configuration between two Martini coarse-grained carbon particles, assuming the real carbon-carbon bond length is 1.2 Å in an $sp^3$ configuration.

-

**[2.5 marks]**

(b) On the provided lattice (Fig. 1), indicate the positions of single Martini carbon particles by drawing circles. The circles should form a lattice that aligns perfectly with the real carbon lattice without leaving any voids or defects. **[2.5 marks]**

**Hint:** Assume that each Martini carbon particle represents four real carbon atoms.

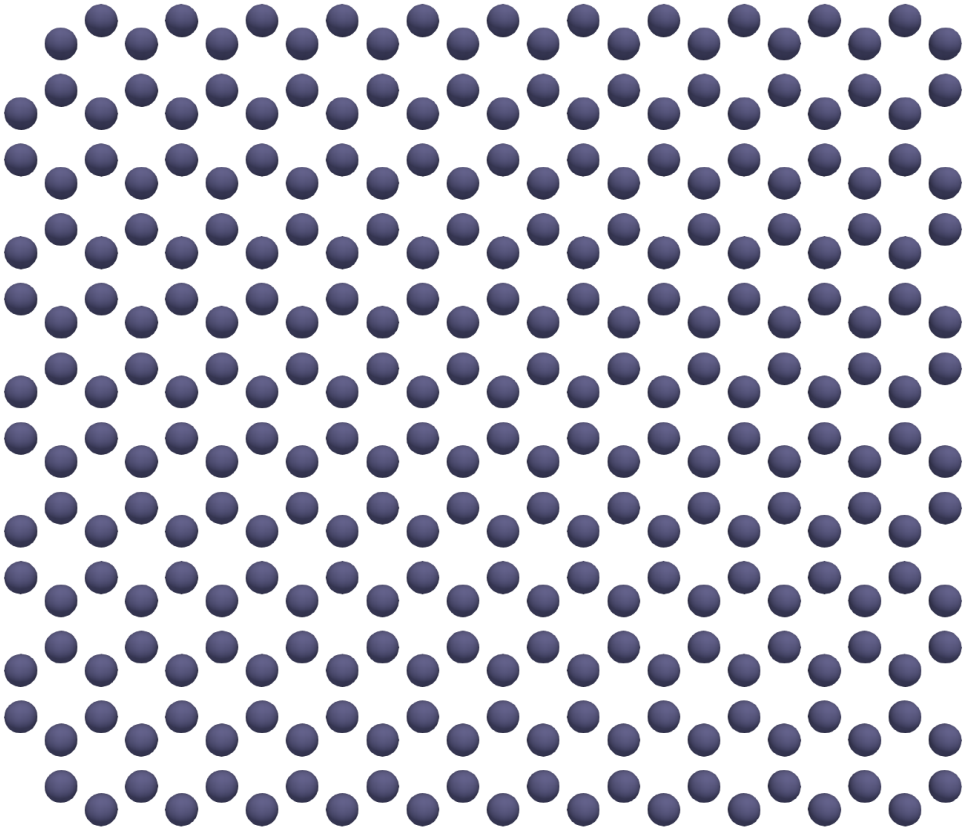


Figure 1: Graphene sheet.

# 3 Manual

## 3.1 Brownian Dynamics and it's characteristics

Use the folder: For exercises: `Exercises/Exercise_1`
LAMMPS input file: `brownian_dynamics.in`

This exercise demonstrates Brownian motion of particles governed by Langevin dynamics and illustrates how diffusive behavior can be quantified through the mean squared displacement (MSD). A collection of non-interacting particles is evolved under stochastic thermal forces, and their trajectories are analyzed to recover diffusive transport properties.

### 3.1.1 System Setup

The following steps describe how the system is constructed and evolved in the LAMMPS input script.

1. **Definition of a large simulation domain**

   ```
   region my_box block 0 30 0 30 0 30
   create_box 1 my_box
   ```

   A cubic simulation box of linear size 30 (in reduced Lennard–Jones units) is created. Periodic boundary conditions are applied in all directions to eliminate surface effects and mimic bulk behavior.

2. **Random initialization of particles**

   ```
   create_atoms 1 random 5000 87910 my_box
   ```

   A total of 5000 particles are randomly distributed throughout the simulation box. This corresponds to a spatially homogeneous initial condition, suitable for studying bulk diffusive motion.

3. **Disabling inter-particle interactions**

   ```
   pair_style none
   ```

   All inter-particle forces are switched off so that particles evolve independently. The resulting dynamics corresponds to ideal Brownian motion, where transport is driven solely by thermal fluctuations and frictional damping.

4. **Initialization of velocities**

```
velocity all create 4.0 12345 mom yes rot no
```

Initial velocities are assigned from a Maxwell–Boltzmann distribution at temperature $k_B T = 4.0$. The total linear momentum is removed to prevent global drift of the system.

5. **Application of Langevin thermostat and time integration**

```
fix thermostat all langevin 4.0 4.0 0.1 89080
fix integrator all nve
```

**Explanation:**

`fix thermostat all langevin 4.0 4.0 0.1 89080`: Applies a Langevin thermostat to all particles, coupling them to an implicit heat bath. The first two parameters (`4.0 4.0`) specify the initial and final target temperatures, which are kept constant throughout the simulation. The damping parameter `0.1` controls the strength of viscous friction: smaller values correspond to weaker coupling (more inertial motion), while larger values lead to overdamped dynamics. The final integer 89080 is a random number seed used to generate the stochastic forces, ensuring reproducibility of the noise sequence.

The Langevin thermostat adds two force contributions to each particle: a deterministic frictional force proportional to velocity, and a random force drawn from a Gaussian distribution. These two terms satisfy the fluctuation–dissipation theorem, ensuring that the system samples the correct canonical ensemble at the specified temperature.

`fix integrator all nve`: Integrates the equations of motion using the velocity-Verlet algorithm. The `nve` integrator updates particle positions and velocities based on the total forces acting on them, including the stochastic and dissipative forces introduced by the Langevin thermostat.

Using `nve` together with `fix langevin` is essential, since the Langevin fix does not perform time integration by itself. Employing an additional thermostatting integrator (such as `nvt`) would result in double thermostatting and unphysical dynamics.

Overall, this combination implements Brownian (stochastic) dynamics at the particle level. At long times and ensemble scales, the resulting trajectories exhibit normal diffusive behavior, with the mean squared displacement growing linearly in time. This microscopic stochastic dynamics corresponds to the macroscopic diffusion equation and is quantitatively described by the Einstein relation between diffusion coefficient, thermal energy, and friction.

6. **Measurement of mean squared displacement**

```
compute msd_all all msd
fix msd_output all ave/time 100 1 100 c_msd_all file msd.dat mode
vector
```

**Explanation:**

`compute msd_all all msd`: This command calculates the mean squared displacement (MSD) of all particles in the system. The MSD is defined as

$$\langle |\mathbf{r}(t) - \mathbf{r}(0)|^2 \rangle,$$

where $\mathbf{r}(t)$ is the particle position at time $t$ and the angular brackets denote an average over all particles. Internally, LAMMPS stores the reference positions (at $t = 0$) and continuously evaluates how far each particle has moved from its initial location.

The MSD is a key dynamical quantity that characterizes particle mobility. For normal diffusion, the MSD grows linearly with time, whereas deviations from linearity indicate subdiffusive or superdiffusive behavior.

`fix msd_output all ave/time 100 1 100 c_msd_all file msd.dat mode vector`: This command time-averages and outputs the MSD data computed by `compute msd_all`. The arguments specify that the MSD is sampled every 100 timesteps, averaged over one sample (no additional smoothing), and written every 100 timesteps to the file `msd.dat`. The `mode vector` option ensures that all components of the MSD (total, $x$, $y$, and $z$) are written to the output file.

The resulting file `msd.dat` contains the time evolution of the MSD. This can be post-processed to extract the diffusion coefficient using the Einstein relation,

$$D = \frac{1}{2d} \frac{d}{dt} \langle |\mathbf{r}(t) - \mathbf{r}(0)|^2 \rangle,$$

where $d$ is the dimensionality of the system.

Overall, this setup enables a direct and quantitative comparison between particle-based Langevin dynamics and the continuum diffusion equation predicted by the Fokker–Planck formalism, thereby linking microscopic stochastic motion to macroscopic transport properties.

### 3.1.2 Instructions

To analyze Brownian motion and extract diffusion coefficient, follow the steps given below:

1. Run the LAMMPS input script `brownian_dynamics.in`.
2. Monitor the simulation output to ensure stable temperature and energy.
3. Open the file `msd.dat` generated during the run.
4. Plot the mean squared displacement as a function of time.
5. Verify that the MSD grows linearly with time, confirming normal diffusion.
6. Extract the diffusion coefficient from the slope of the MSD curve using the Einstein relation.

Note: Because inter-particle interactions are disabled, the observed diffusive behavior arises purely from the stochastic forces introduced by the Langevin thermostat. Similar diffusion can be obtained using other stochastic integrators, provided the fluctuation-dissipation balance is maintained.

### 3.1.3 Learning Outcome

After completing this exercise, students will be able to:

1. Understand how Langevin dynamics models Brownian motion through stochastic and dissipative forces.
2. Set up particle-based simulations of ideal diffusion in the absence of inter-particle interactions.
3. Recognize the physical meaning of the mean squared displacement and its role in characterizing diffusion.
4. Compute and analyze MSD data from molecular dynamics simulations.
5. Verify the linear time dependence of MSD for normal diffusion.
6. Extract diffusion coefficients from simulation data using the Einstein relation.
7. Relate microscopic stochastic particle motion to macroscopic diffusive transport.
8. Develop practical skills in combining LAMMPS simulations with post-processing and data analysis.

## 3.2 Studying the self-assembly in patchy-particle model

Use the Folder: `Hands_on/patchy_particle_model`, For exercises: `Exercises/Exercise_2`
LAMMPS input file: `patchy_particle.in`, LAMMPS input data file: `patchy_molecule.mol`

Patchy particle models are coarse-grained representations of complex molecules in which interactions are directionally selective rather than isotropic. Each particle consists of a central core decorated with a finite number of attractive interaction sites (*patches*) located at specific positions on its surface. These patches introduce anisotropy into the interaction potential, allowing particles to preferentially bond in certain orientations.
Such models are widely used to mimic systems where directional bonding plays a crucial role, including self-assembling colloids, protein-like particles, hydrogen-bonded fluids, and supramolecular polymers. By controlling the number, arrangement, and strength of patches, one can tune the resulting structures, ranging from linear chains and branched networks to crystalline or gel-like phases.
In this tutorial, the goal is to perform molecular dynamics simulations of a patchy particle system using a constant-temperature thermostat (NVT ensemble). By varying the temperature, students will study how thermal fluctuations compete with directional attractions, leading to distinct phase behaviors. In particular, the simulations aim to identify temperature regimes where particles form transient or persistent chains, exhibit liquid-like aggregation, or remain in a dispersed, gas-like state.
Through this exercise, students will gain insight into how anisotropic interactions govern self-assembly and phase behavior, and how simplified coarse-grained models can capture essential physical mechanisms underlying complex soft-matter systems.

### 3.2.1 System Setup

**Molecular Template Construction**

1. Molecule template can have all the information regarding a single molecules. A number of attributes and their values can be specified in the molecular template e.g., number of atoms, their corresnding coordinates, atom types, number of bonds bond types and atoms involved in bonds with the bond id. Similary it can also be specified by its full topology including bond angle, dihedral, improper dihedral etc. Usually a simple molecular template with no-bond topology looks like this.

```
5 atoms

Coords

1 1.0 1.0 1.0
2 1.2886 1.2886 1.2886
3 0.7113 0.7113 1.2886
4 0.7113 1.2886 0.7113
5 1.2886 0.7113 0.7113

Types

1 1
2 2
3 2
4 2
5 2
```

An incorrect formate or missing coordinate or particle type specification can lead to an error in LAMMPS.

**LAMMPS input**

1. **Defining essential variables and simulation units**

```
units lj
atom_style hybrid sphere molecular
boundary p p p
pair_style hybrid lj/cut 2.0 cosine/squared 0.12
neighbor 0.3 bin
neigh_modify every 1 delay 0 check yes
```

**Explanation:**

`units lj`: Defines reduced Lennard-Jones units, which simplifies the simulation by normalizing mass, length, energy, and time. This is standard for coarse-grained Brownian dynamics and ensures numerical stability.

`atom_style hybrid sphere molecular`: Allows the system to include spherical cores (for Brownian particles) and additional molecular structures (patches) in the same simulation. This hybrid style is essential for modeling patchy particles with directional interactions.

`boundary p p p`: Periodic boundaries in x, y, and z directions create an effectively infinite system, preventing surface effects and mimicking bulk fluid behavior.

`pair_style hybrid lj/cut 2.0 cosine/squared 0.12`: Combines two types of interactions: `lj/cut 2.0` handles core-core repulsions via Lennard-Jones potential with a cutoff

of 2.0. `cosine/squared 0.12` models directional patch-patch interactions using a Gaussian-like cosine-squared potential. The cutoff values (2.0 and 0.12) are chosen to balance computational efficiency and physical accuracy.

`neighbor 0.3 bin`: Sets the neighbor list skin distance for the Verlet neighbor list. Particles within this distance are considered neighbors for force calculations, improving efficiency.

`neigh_modify every 1 delay 0 check yes`: Updates the neighbor list every timestep with no delay and enables consistency checking. This is important for Brownian dynamics, where particles move stochastically, to avoid missing interactions.

**Additional notes:** These settings form the foundation for accurate Brownian dynamics, ensuring the simulation respects both thermal fluctuations (via Langevin thermostat later) and directional interactions between patches. - Careful choice of units and neighbor parameters directly affects simulation stability, time-step selection, and correct reproduction of diffusive and self-assembly behavior.

2. **Creating the simulation box**

```
region box block 0 30.0 0 30.0 0 30.0
create_box 2 box
```

**Explanation:**

`region box block 0 30.0 0 30.0 0 30.0`: Defines a cubic simulation box extending from 0 to 30 units in x, y, and z directions. This provides the spatial domain for particles to move.

`create_box 2 box`: Creates the simulation box with **2 particle types**. Type 1 represents the Brownian core particles, and type 2 represents the patches that mediate directional interactions.

Choosing these box dimensions ensures a sufficient number of particles for meaningful self-assembly, while avoiding overly sparse systems that would slow down equilibration.

This box setup is compatible with **periodic boundary conditions** defined earlier, which mimic an infinite bulk system without surface effects.

3. **Loading molecular template and creating particles**

```
molecule patchy_part patchy_molecule.mol
create_atoms 0 random 1000 87910 NULL mol patchy_part 454756 overlap
1.5 maxtry 50
```

**Explanation:**

`molecule patchy_part patchy_molecule.mol`: Loads the molecular template defining the patchy particle structure. Each particle has a core and one or more patches that enable directional interactions.

`create_atoms 0 random 1000 87910 NUL ...`: Creates **1000 molecules** randomly in the simulation box. **0** specifies the region ID (here the entire box). **87910** is the user-

defined random seed ensuring reproducibility. **overlap 1.5** ensures that newly created particles are not placed too close to existing particles. **maxtry 50** limits the number of attempts to place each molecule without overlaps. **454756** is an additional seed for placement of molecular atoms.

This setup ensures that patchy particles are distributed randomly without initial overlaps, which helps the system equilibrate faster and form self-assembled structures efficiently.

-By separating the core and patches using the molecular template, rigid-body constraints can later maintain the proper core-patch geometry during Brownian dynamics.

4. **Defining particle properties and interactions**

```
pair_coeff 1 1 lj/cut 0.01 1.3 2.0
pair_coeff 1 2 none
pair_coeff 2 2 cosine/squared 8 0.3 0.35

set type 1 mass 1.0
set type 2 mass 0.000001

set type 1 diameter 1.0
set type 2 diameter 0.0
```

**Explanation:**

`pair_coeff 1 1 lj/cut 0.01 1.3 2.0`: Defines Lennard-Jones interactions between core particles (type 1). `0.01` is the interaction strength ($\epsilon$), `1.3` is the particle diameter ($\sigma$), `2.0` is the cutoff distance.

`pair_coeff 1 2 none`: Core-patch interactions are ignored. Patches only interact via directional forces.

`pair_coeff 2 2 cosine/squared 8 0.3 0.35`: Patch-patch interactions use a Gaussian-like cosine-squared potential: `8` is the strength of the directional bond, `0.3` is the equilibrium distance, `0.35` is the interaction width.

`set type 1 mass 1.0` and `set type 2 mass 0.000001`: Core particles are massive; patches are almost massless to enforce rigid geometry without influencing dynamics significantly.

`set type 1 diameter 1.0` and `set type 2 diameter 0.0`: Core diameter defines the main excluded volume; patch diameter is negligible to allow directional bonding.

This configuration ensures: Cores maintain physical volume. Patches control bonding directionality without affecting Brownian motion significantly. Interactions are physically meaningful and numerically stable for self-assembly simulations.

5. **Grouping particles and neighbor exclusions**

```
group core type 1
group patch type 2
group rigid_molecule type 1 2
neigh_modify exclude molecule/intra rigid_molecule every 1 delay 0
check no
```

**Explanation:** - Groups let you apply fixes selectively (e.g., thermostat to cores, rigid integration to molecules).

`neigh_modify exclude molecule/intra rigid_molecule`: skips intramolecular force calculations for efficiency.

6. **Applying thermostat and rigid-body integration**

```
fix thermo_stat core langevin 1.0 1.0 0.1 428984 omega yes
fix rigid_thermo rigid_molecule rigid/small molecule
```

**Explanation:**

`fix thermo_stat core langevin 1.0 1.0 0.1 428984 omega yes`: Applies a Langevin thermostat to the core particles, maintaining the target temperature.

Models stochastic Brownian dynamics by introducing random forces and damping consistent with the fluctuation-dissipation theorem. `1.0 1.0` are the initial and target temperatures, `0.1` is the damping parameter, `428984` is the random seed, `omega yes` includes rotational degrees of freedom for spherical particles.

`fix rigid_thermo rigid_molecule rigid/small molecule`: Integrates core and patch particles as a rigid body, preserving bond lengths and relative orientation. Ensures the patch remains correctly attached to the core while the molecule undergoes Brownian motion.

This setup ensures: Core particles follow Langevin dynamics simulating Brownian motion. Rigid molecules maintain structural integrity. Thermal fluctuations drive self-assembly without distorting particle geometry.

7. **Dumping trajectories for visualization**

```
dump 1 all custom 10 simulation_data.LAMMPStrj id type x y z mol
```

**Explanation:**

`dump 1 all custom 10 simulation_data.LAMMPStrj id type x y z mol`: Creates a trajectory file `simulation_data.LAMMPStrj` for all particles. `1` is the dump ID, `all` specifies all particles are included, `custom` allows specifying which properties to record, `10` is the interval (every 10 timesteps). Recorded properties: `id`: particle ID, `type`: particle type (core or patch), `x y z`: particle positions, `mol`: molecule ID for grouping rigid bodies.

This setup ensures: Trajectories can be visualized in OVITO or other post-processing tools. Enables analysis of self-assembly patterns, cluster formation, and particle dynamics. Provides data for calculating structural and dynamical properties such as MSD, radial distribution functions, or aggregation statistics.

8. **Computing kinetic energy and temperature of core particles**

```
compute kinetic_core core ke
fix kinetic_output core ave/time 100 1 100 c_kinetic_core file
kinetic.dat mode scalar
compute temp_core core temp/sphere
fix temp_output core ave/time 100 1 100 c_temp_core file
temperature.dat mode scalar
```

**Explanation:**

`compute kinetic_core core ke`: Calculates the kinetic energy of core particles only.

`fix kinetic_output core ave/time 100 1 100 c_kinetic_core file kinetic.dat mode scalar`: Averages kinetic energy over 100 steps and writes to `kinetic.dat`. Allows monitoring of system equilibration and temperature fluctuations.

`compute temp_core core temp/sphere`: Computes the temperature of core particles using spherical degrees of freedom.

`fix temp_output core ave/time 100 1 100...` : Outputs the averaged core temperature to `temperature.dat`. Helps verify that Brownian dynamics and thermostatting maintain the target temperature. This setup ensures that rigid patches do not interfere with the thermal properties of the core particles, maintaining proper stochastic dynamics.

9. **Time step and thermodynamic output settings**

```
timestep 0.005
thermo 100
thermo_style custom step temp ke pe press c_kinetic_core c_temp_core
```

**Explanation:**

`timestep 0.005`: Small timestep ensures numerical stability for rigid core-patch molecules and accurate stochastic Brownian motion integration. `thermo 100`: Prints thermodynamic output every 100 steps for monitoring system behavior. `thermo_style custom ...`: Specifies which quantities to output (step, temperature, kinetic energy, potential energy, pressure, and core-specific computed values). This setup allows continuous monitoring of energy, temperature, and pressure to verify stability and correctness of the simulation.

10. **Running the simulation**

```
run 100000
```

### 3.2.2 Instructions

System evolution and equilibrium structure can be initially verified by visualization. To visualize the system in OVITO, you can follow the following steps.

1. Load the data file simulation_data.LAMMPStrj in ovito.
2. Visualize the particles, which might show gas or liquid like structure e.g, globules, chains etc.
3. Change the size and color of the particles so that different particles are distinguishable by their particle type.
4. Take the screenshot produce, it as a simplest evidence of the given phase e.g, liquid, gel.

Note: In research, as an standard practice, we look at the evolution of the potential energy. If with time, it is changing, it means the total energy of the system has still not reached the equilibrium condition. For an ergodic system where time average becomes equal to the ensemble average, the stagnation of total energy and therefore essentially the potential energy can be a reliable basis to consider the state of the system in equilibrium. The potential energy evolution can be seen in the LAMMPS file printed in the column `pe`.

### 3.2.3 Learning Outcomes

After completing the steps in this section, students will be able to:

1. How to setup a LAMMPS script to study the self-assembly using patchy particle model.
2. Understand how to load and visualize particle-based simulation data in OVITO.
3. Identify structural features of different phases (gas, liquid, or mixed) by observing particle arrangements such as clusters, chains, or globules.
4. Interpret the evolution of potential energy (`pe`) over time as a criterion for equilibrium.
5. Develop an intuitive understanding of system equilibration, including the significance of energy fluctuations and the approach to steady-state behavior.

## 3.3 Viscosity of a Lennard-Jones Fluid Using the Green-Kubo Relation

Use the Folder: For exercises: `Exercises/Exercise_3`
LAMMPS input file: `viscosity_input.in`

In this exercise, the shear viscosity of a Lennard–Jones (LJ) fluid is computed using equilibrium molecular dynamics and the Green–Kubo formalism. The viscosity is obtained from the time integral of the autocorrelation function of the off-diagonal components of the pressure tensor.
The shear viscosity is defined as

$$\eta = \frac{V}{k_B T} \int_0^\infty \langle P_{\alpha\beta}(0)\, P_{\alpha\beta}(t) \rangle \, dt, \tag{1}$$

where $P_{\alpha\beta} \in \{P_{xy}, P_{xz}, P_{yz}\}$, $V$ is the system volume, and $T$ is the temperature.

### 3.3.1 System setup

1. **Simulation units and general settings**

```
units lj
dimension 3
boundary p p p
atom_style atomic
```

**Explanation:**

`units lj`: Enables reduced Lennard–Jones units, where the fundamental quantities are nondimensionalized such that the particle diameter $\sigma = 1$, interaction strength $\epsilon = 1$, particle mass $m = 1$, and Boltzmann constant $k_B = 1$. This choice simplifies the equations of motion and allows results to be expressed in universal, system-independent form.

`dimension 3`: Specifies that the simulation is performed in three spatial dimensions, which is required for correctly modeling bulk fluid behavior and transport properties such as viscosity.

`boundary p p p`: Applies periodic boundary conditions in the $x$, $y$, and $z$ directions. This removes surface effects and mimics an infinite homogeneous system by allowing particles exiting one side of the simulation box to re-enter from the opposite side.

`atom_style atomic`: Defines particles as structureless point atoms without internal degrees of freedom. This is appropriate for monatomic Lennard–Jones fluids and ensures that the computed stress tensor and velocity correlations arise solely from translational motion and interparticle forces.

2. **User-defined thermodynamic and numerical parameters**

```
variable T equal 1.0
variable Tinit equal 1.2
variable dt equal 0.00001
```

**Explanation:**

`variable T equal 1.0`: Defines the target temperature for the production phase of the simulation. This value sets the thermal energy scale in reduced Lennard–Jones units and is used by the thermostat to maintain equilibrium conditions.

`variable Tinit equal 1.2`: Specifies a slightly higher initial temperature used during early equilibration. Starting at a higher temperature helps particles overcome unfavorable initial configurations and accelerates relaxation toward equilibrium.

`variable dt equal 0.00001`: Defines a very small integration timestep. This is particularly important during the initial relaxation phase, where large forces or overlaps may occur, and ensures numerical stability before switching to a larger production timestep.

3. **Stress autocorrelation sampling parameters**

```
variable p equal 400
variable s equal 5
variable d equal $p*$s
```

**Explanation:**

`variable p equal 400`: Defines the number of correlation points stored for each component of the stress tensor. This parameter controls the maximum time lag over which stress correlations are evaluated.

`variable s equal 5`: Sets the number of timesteps between successive stress samples. Smaller values increase temporal resolution but also increase data storage and computational cost.

`variable d equal $p*$s`: Defines the total correlation length in timesteps over which the Green-Kubo stress autocorrelation function is computed. This determines the integration window used to calculate transport coefficients such as viscosity.

These parameters together control the temporal extent and resolution of stress autocorrelation sampling and must be chosen sufficiently large and also sufficiently resolved to ensure that correlations decay to zero, guaranteeing reliable Green–Kubo integrals.

4. **System setup and density control**

```
region box block 0 20 0 20 0 20
create_box 1 box
lattice fcc 0.6
create_atoms 1 box
mass 1 1.0
```

**Explanation:**

`region box block 0 20 0 20 0 20`: Defines a cubic simulation domain extending from 0 to 20 in the $x$, $y$, and $z$ directions. This choice sets the system volume and directly controls the particle number density when combined with the lattice spacing.

`create_box 1 box`: Creates a simulation box containing one particle type, sufficient for a single-component Lennard–Jones fluid.

`lattice fcc 0.6`: Initializes atoms on a face-centered cubic lattice with a reduced lattice density of 0.6. Using an FCC lattice provides a dense but well-ordered initial configuration, while the reduced density helps prevent large initial overlaps.

`create_atoms 1 box`: Populates the simulation box with atoms placed at the lattice sites defined by the FCC lattice. This ensures a uniform initial particle distribution.

`mass 1 1.0`: Assigns unit mass to all particles, consistent with reduced Lennard–Jones units.

A careful choice of lattice density and box size is essential to achieve the desired number density while avoiding excessive overlaps across periodic boundaries, which could otherwise lead to large initial forces and numerical instabilities.

5. **Interatomic interactions**

```
pair_style lj/cut 3.5
pair_modify shift yes
pair_coeff 1 1 1.0 1.0 3.5
```

**Explanation:**

`pair_style lj/cut 3.5`: Specifies the Lennard–Jones pair potential truncated at a cutoff distance of $3.5\sigma$. This cutoff is sufficiently long to capture attractive interactions while remaining computationally efficient.

`pair_modify shift yes`: Shifts the Lennard–Jones potential so that it smoothly goes to zero at the cutoff distance. This removes discontinuities in the potential energy, which is essential for accurate force and pressure calculations.

`pair_coeff 1 1 1.0 1.0 3.5`: Defines the interaction parameters between particles of type 1: $\epsilon = 1.0$ sets the interaction strength, $\sigma = 1.0$ sets the characteristic particle diameter, and 3.5 is the cutoff radius.

Using a smoothly truncated potential is crucial for reliable evaluation of the pressure tensor and its time autocorrelation, which directly enters the Green–Kubo calculation of viscosity.

6. **Time integration and thermodynamic output**

```
timestep $dt
thermo $d
thermo_style custom step temp press ke
```

**Explanation:**

`timestep $dt`: Sets the simulation timestep using the previously defined variable `dt`. This ensures consistent time integration across different stages of the simulation.

`thermo $d`: Controls the frequency of thermodynamic output so that it is synchronized with the stress autocorrelation sampling window. This allows direct monitoring of system stability over the full Green–Kubo integration interval.

`thermo_style custom step temp press ke`: Specifies the thermodynamic quantities printed to the log file, including timestep number, instantaneous temperature, pressure, and kinetic energy. These outputs are used to verify energy conservation and equilibrium behavior during the simulation.

7. **Equilibration phase**

```
velocity all create $Tinit 12345 dist gaussian
velocity all scale $T
fix thermostat all langevin $T $T 0.1 34300
fix integrator all nve
run 100000
```

**Explanation:**

`velocity all create $Tinit 12345 dist gaussian`: Initializes particle velocities from a Maxwell–Boltzmann distribution at the higher initial temperature `Tinit`. The random seed ensures reproducibility of the velocity initialization.

`velocity all scale $T`: Rescales velocities to match the target temperature `T`, providing a smooth transition toward the desired equilibrium state.

`fix thermostat all langevin $T $T 0.1 34300`: Applies a Langevin thermostat to all particles, introducing stochastic and damping forces consistent with the fluctuation–dissipation theorem. This accelerates thermal equilibration while maintaining the correct temperature.

`fix integrator all nve`: Integrates the equations of motion using the microcanonical (NVE) scheme while the thermostat controls temperature during equilibration.

`run 100000`: Runs the equilibration phase for a sufficient number of timesteps to allow the system to relax structurally and thermodynamically.

Thermostats are appropriate during equilibration to reach equilibrium efficiently, but they must be removed before Green–Kubo production runs to avoid biasing equilibrium stress fluctuations.

8. **Production run and stress autocorrelation**

```
fix integrator all nve
reset_timestep 0
variable pxy equal pxy
variable pxz equal pxz
variable pyz equal pyz
fix SS all ave/correlate $s $p $d v_pxy v_pxz v_pyz type auto file
SOS.dat ave running
```

**Explanation:**

`fix integrator all nve`: Performs microcanonical (NVE) integration, ensuring energy-conserving dynamics required for accurate Green–Kubo viscosity calculations.

`reset_timestep 0`: Resets the timestep counter to zero at the start of the production run, helpful for data analysis and correlation function alignment.

`variable pxy/pxz/pyz equal ...`: Stores the instantaneous off-diagonal components of the pressure tensor ($P_{xy}, P_{xz}, P_{yz}$) for later correlation analysis.

`fix SS all ave/correlate ...`: Computes the time autocorrelation functions of the off-diagonal pressure tensor components over the specified sampling window (`s`, `p`, `d`). These autocorrelations are the microscopic fluctuations whose time integral yields the shear viscosity via the Green–Kubo relation.

This setup ensures that stress fluctuations are properly sampled without thermostat interference, providing the core data for transport coefficient evaluation.

9. **Green–Kubo integration and viscosity output**

```
variable V equal vol
variable cyan equal $V/($kB*$T)*$s*$dt
variable v11 equal trap(f_SS[3])*$scale
variable v22 equal trap(f_SS[4])*$scale
variable v33 equal trap(f_SS[6])*$scale
variable eta equal (v_v11+v_v22+v_v33)/3.0
```

**Explanation:**

`variable V equal vol`: Stores the instantaneous system volume for scaling the Green–Kubo integral.

`variable scale equal ...`: Calculates the prefactor $\frac{V}{k_B T}\Delta t$ used to convert the discrete stress autocorrelation sums into viscosity in reduced LJ units. Here, `$V`, `$T`, `$kB`, `$dt`, `$s` are previously defined variables controlling volume, temperature, Boltzmann constant, timestep, and sampling interval.

`variable v11/ v22/ v33 equal trap(f_SS[3..5]*scale)`: Performs trapezoidal numerical integration of the autocorrelation functions for $P_{xy}$, $P_{xz}$, and $P_{yz}$, yielding partial viscosity contributions.

`variable eta equal (v_v11+v_v22+v_v33)/3.0`: Averages the three off-diagonal components to obtain the final shear viscosity, improving statistical accuracy.

This setup ensures that the Green-Kubo integral is computed correctly, producing reliable viscosity estimates from equilibrium stress fluctuations. Beyond this exercise also, setting up scale variables and sampling duration, average and timne resolution (timestep) is one of the most crucial steps to ensure that the computed viscosity is very close to the actual value.

### 3.3.2 Instructions

All the calculations are done in the LAMMPS input for the calculation of viscostiy using Green-Kubo relation. Now, the viscosity can be obtained following the instructions given below.

1. Choose the given density.
2. Adjust the box size between 20-30 and calibrate until you stop getting lost atoms or segmentation fault.
3. Choose the appropriate time steps for the equilibration. Remember if you will chose the too large time steps it will blow up the simulation. If too small then system will not equilibrate, within the given simulation time.
4. Run the simulation and wait until finished.
5. Open the LAMMPS' log file and scroll down to check the printed value of viscosity.
6. Follow the same strategy for the different density.

**Note:** Proper selection of simulation parameters e.g, time steps, equilibration period, is crucial for obtaining reliable transport properties and avoiding simulation artifacts in numerical results, e.g, getting zero viscosity for a dilute system.

### 3.3.3 Learning Outcomes

After completing the steps in this section, students will be able to:

1. Set up LAMMPS simulations for viscosity calculations using the Green-Kubo formalism.
2. Select appropriate system parameters such as density, box size, and time step to ensure stable and accurate simulations.
3. Identify and troubleshoot common simulation issues such as lost atoms or segmentation faults.
4. Understand the importance of equilibration time and time step selection on the accuracy and stability of molecular dynamics simulations.

5. Extract and interpret viscosity values from LAMMPS log files.
6. Analyze the effect of varying density on the computed viscosity and relate the results to physical behavior.
7. Develop skills in performing systematic parametrization and troubleshooting often faced in simulations of complex systems.

## 3.4 Calculation of liquid-vapor interface in Lennard-Jones (LJ) fluid

Use the folder: For exercises: `Exercises/Exercise_4`
LAMMPS input file: `liquid_box_creator.in`, `intermediate.in`, `coexistence_simulator.in`

In this section, you will calculate the coexistence density between the gas and liquid phases of a Lennard-Jones fluid. The aim is first to create a highly dense liquid phase without any bubbles or nuclei of the gas phase, then merge the liquid phase with a gas phase. Finally, you will calculate the density profile of the combined system after equilibrating it.
Multiple LAMMPS input files are used to achieve this, the relevant section in each of the files are described below.

### 3.4.1 System Setup

To make a high-density liquid regime, you will use the file `liquid_box_creator.in`. Key commands are described below:

1. **Creating a large rectangular box:**

```
region box block 0 15 0 15 0 90
create_box 1 box
```

   All details of box creation have been explained previously. This initializes a rectangular simulation domain with one particle type.

2. **Applying NpT ensemble to compress the fluid:**

```
fix barostat all npt temp 0.4 0.4 1 z 10 10 1.0 couple none
```

   **Explanation:**

   The NpT ensemble is used to control temperature and pressure. Temperature is set to `0.4 0.4`, and a barostat is applied along $z$ with pressure `10 10`. Pressure along only the $z$-axis controls the box height; the shape along $x$ and $y$ remains unconstrained. This compresses the box into a high-density liquid configuration.

3. **Writing the final configuration:**

```
write_data compressed_coordinate.data
```

   **Explanation:**

Saves the equilibrated particle positions. The box is compressed along $z$ and typically located near the center of the original box (e.g., spanning $38-52$ if the original box spanned $0-90$).

### Merging Liquid and vapor Phases

Now we want to compressed liquid to extend in a larger box, e.g., spanning $20-70$ along $z$. So that the empty regions ($20-38$ and $52-70$) can be filled with low-density gas particles.

### Removing periodic boundary condition

To get rid of the periodic boundary condition in one direction. We can reload the output of the first LAMMPS script and apply the following command run for zero time step. A few of the key commands are mentioned below.The relevant file in this section is intermediate.in.

1. **Removing periodicity along the z-axis**

```
boundary p p s
```

**Explanation:**

The `s` flag disables periodic boundaries along $z$. This ensures that walls fully enclose all particles along $z$, avoiding wrapping interactions. Updated data is saved and later loaded for coexistence simulation.

### Filling up vapor particles

Now we have periodic boundary condition removed and the data is exported to an output file (see the instructions section). To fill the vapor particle we need to chose two regions on both sides of the liquid region and then create the new atoms, which can be done by the following command. The relevant file is coexistence_simulator.in

1. **Defining gas regions and adding particles**

```
region gas_region_1 block 0 15 0 15 20 35
region gas_region_2 block 0 15 0 15 55 70
create_atoms 1 random 100 87910 gas_region_1 overlap 1.0 maxtry 50
create_atoms 1 random 100 87910 gas_region_2 overlap 1.0 maxtry 50
```

Two regions within the simulation box are defined and filled with gas particles at random positions.

2. **Binning particle numbers along the z-axis**

```
compute chunk_1 all chunk/atom bin/1d z lower 0.02 units reduced
compute myChunk1 all property/chunk chunk_1 count
fix 1 all ave/time 100 1 100 c_myChunk1 ...  file
bin_particles.LAMMPStrj mode vector
```

Divides the $z$-axis into bins of size 0.02 (reduced units). Counts the number of particles in each bin and outputs to a file for later density calculations.

### 3.4.2 Instructions

1. Run `liquid_box_creator.in` to generate the given high-density liquid. Proper liquid density prevents bubble growth during gas-liquid simulation.
2. Run `intermediate.in` to remove periodicity along the $z$-axis from `compressed_coordinate.data`.
3. Open the data file and extend the $z$-axis symmetrically, e.g., change `38 52 zlo zhi` to `20 70 zlo zhi`. Save the file.
4. Run `coexistence_simulator.in` to equilibrate the system and record particle counts in bins along $z$. Output is stored in `bin_particles.LAMMPStrj`.
5. Run `data_rectifier.py` to average particle counts over time and produce the density profile $\rho(z)$ in `averaged_density_profile.txt`.
6. Open DESMOS or any online plotter and type $tanh(x)$ to observe the characteristic interface shape, similar to the gas-liquid interface.
7. Run `density_profile_fitter.py` to fit $\rho(z)$ to a $tanh$ function. The script outputs gas and liquid densities and a plot `density_profile_fit.png`.
8. Change the temperature of all of your LAMMPS input script and follow the same procedure to get the coexistence at that temperature **(Not needed for the exercise)**.
9. Record coexistence densities: first column for temperature, second for vapor density, third for liquid density. Save as `liquid_vapor_densities.txt` **(Not needed for the exercise)**.
10. Run `plotter.py` to generate the Binodal plot showing the coexistence curve and critical point, **(Not needed for the exercise)**.

Note: This exercise emphasizes the link between microscopic particle behavior and macroscopic thermodynamic properties, illustrating how molecular simulations can predict phase coexistence and critical phenomena. Therefore, for the research purpose system size effect must be checked.

### 3.4.3 Learning Outcome

After completing this exercise, students will be able to:

1. Understand the procedure to generate a high-density liquid phase and prevent unwanted bubble formation during gas-liquid simulations.
2. Learn how to remove periodic boundary conditions along a selected axis and manipulate the simulation box to accommodate multi-phase systems.
3. Comprehend how to merge liquid and gas phases and equilibrate the combined system to reach steady-state coexistence.
4. Apply spatial binning to compute particle number density along the interface and analyze density profiles.
5. Develop practical skills in integrating LAMMPS simulations with post-processing tools for visualization, fitting, and plotting.

## 3.5 Shear Viscosity of a Lennard-Jones Fluid Using Non-Equilibrium MD

Use the folder: `Hands_on/viscosity_through_shearing`
LAMMPS input file: `lj_viscosity_shearing.in`

This exercise computes the shear viscosity of a Lennard-Jones (LJ) fluid using non-equilibrium molecular dynamics (NEMD). This is one of the simplest example of non-equilibrium molecular dynamics. A steady shear flow is induced by moving solid wall regions, and the viscosity is obtained from the resulting velocity gradient and shear stress.
The simulations are performed using the folder `viscosity_through_shearing`, with the input file `lj_viscosity_shearing.in` and analysis via the Python script `viscosity_analysis.py`.

### 3.5.1 Simulation setup

1. **Defining the simulation domain and wall regions**

```
region box block -3 23 0 20 0 20
create_box 3 box
region lowersec block INF 0 INF INF INF INF
region uppersec block 20 INF INF INF INF INF
```

**Explanation:**

`region box block -3 23 0 20 0 20`: Defines the overall simulation box using a cuboidal region. The $x$-dimension is extended beyond the fluid domain ($-3$ to 23) to allow space for wall regions. The $y$ and $z$ dimensions (0–20) define the lateral size of the system. Choosing slightly larger boundaries in the shear-gradient direction avoids boundary artifacts and ensures that the wall regions can move without interfering with periodic images of the fluid.

`create_box 3 box`: Creates a simulation box containing three distinct atom types: one for the fluid and two for the walls. This separation allows independent control of wall and fluid behavior (e.g., wall velocity, freezing, thermostatting). Specifying the box after defining the region ensures that all subsequent atoms are placed within the correct spatial domain.

`region lowersec block INF 0 INF INF INF INF`: Defines the lower wall slab. Here, `INF` indicates unbounded in $x$, $z$, or other directions not specified, while $y = 0$ is the lower boundary of the wall. This creates a thin slab at the bottom that can be treated as a rigid wall during shear.

`region uppersec block 20 INF INF INF INF INF`: Defines the upper wall slab starting at $x = 20$ extending to infinity. Like the lower wall, this region is isolated from the central fluid. The placement at the opposite end of the shear-gradient direction enables the generation of a uniform shear flow when the wall moves.

**Key points:** The central region between the lower and upper walls becomes the fluid domain where viscosity is measured. Using separate regions for walls allows selective application of velocity, force constraints, and thermostats without affecting the fluid. Extending boundaries beyond the fluid ensures walls do not overlap or introduce artifacts due to periodic boundary conditions. Proper definition of wall thickness and placement is essential for generating a stable, realistic shear profile in NEMD simulations.

2. **Creating particles and defining wall and fluid groups**

```
lattice fcc 0.3
create_atoms 1 box
group upper region uppersec
group lower region lowersec
group wall union upper lower
group fluid subtract all wall
set group upper type 2
set group lower type 3
```

**Explanation:**

`lattice fcc 0.3`: Initializes a face-centered cubic (FCC) lattice with spacing 0.3 in reduced units. The lattice spacing controls the number density of particles, ensuring that the fluid has the desired density and that wall particles are properly packed for mechanical stability.

`create_atoms 1 box`: Populates the simulation box with particles of type 1 (fluid atoms) on the defined FCC lattice. This provides a uniform, overlap-free starting configuration for the system.

`group upper/lower region uppersec/lowersec`: Creates groups for the wall regions. Particles in the `uppersec` and `lowersec` slabs are assigned to the `upper` and `lower` groups, respectively. Grouping allows selective manipulation, e.g., applying velocities, freezing, or assigning different atom types.

`group wall union upper lower`: Combines the upper and lower wall groups into a single `wall` group. This makes it easy to apply constraints, forces, or thermostats uniformly to all wall particles.

`group fluid subtract all wall`: Defines the `fluid` group as all atoms not in the wall. This ensures that only the central fluid particles are subject to thermostatting and shear flow measurement.

`set group upper/lower type 2/3`: Assigns distinct atom types to the upper (type 2) and lower (type 3) walls. Different types allow the application of different velocities, forces, or interaction parameters if needed, e.g., moving the upper wall to generate shear while keeping the lower wall stationary.

**Key points:**

Separation of wall and fluid particles is crucial for imposing shear without disturbing the bulk fluid. Distinct atom types for walls allow independent control over motion and inter-

action parameters. Creating groups ensures that fixes (thermostats, velocity assignments, setforce) can be applied selectively, preserving the integrity of fluid dynamics and shear profile measurements. FCC lattice initialization prevents particle overlap and provides a reproducible starting configuration for NEMD simulations.

3. **Inter-particle interactions and integration scheme**

```
pair_style lj/cut 3.0
pair_coeff * * 1.0 1.0 3.5
pair_modify shift yes
fix integrator all nve
```

**Explanation:**

All particles interact via a truncated Lennard-Jones potential ($r_c = 3.0$). The potential is shifted to zero at the cutoff for smooth energy and force evaluation. The `nve` integrator evolves particle positions and velocities without explicit thermostatting, as temperature control is applied separately to the fluid.

4. **Applying shear and thermostatting the fluid**

```
velocity fluid create $T 12345 mom yes rot yes
velocity upper set 0.0 $srate 0.0 sum no
compute bias fluid temp/partial 1 0 1
fix frozenwall wall setforce 0.0 0.0 0.0
fix thermostat fluid langevin 1.1 1.1 0.1 933888
fix_modify thermostat temp bias
```

**Explanation:**

`velocity fluid create $T 12345 mom yes rot yes`: Initializes the velocities of fluid particles according to a Maxwell–Boltzmann distribution at the target temperature $\$T$. The random seed (12345) ensures reproducibility. The options `mom yes` and `rot yes` remove net linear and angular momentum, preventing drift or rotation of the entire fluid block.

`velocity upper set 0.0 $srate 0.0 sum no`: Imposes a constant velocity $\$srate$ along the $y$-direction to the upper wall particles. This generates shear flow across the fluid, while `sum no` ensures that velocities are set individually without altering the total momentum of the group.

`compute bias fluid temp/partial 1 0 1`: Calculates the thermal temperature of the fluid after subtracting contributions from streaming motion in the shear-gradient ($x$) and flow ($y$) directions. This ensures that thermostatting only affects random thermal fluctuations, not the imposed shear.

`fix frozenwall wall setforce 0.0 0.0 0.0`: Freezes wall particles by removing all forces, keeping them stationary or moving only due to explicitly assigned velocities. This maintains a rigid boundary for shear generation.

`fix thermostat fluid langevin 1.1 1.1 0.1 933888`: Applies a Langevin thermostat to the fluid to maintain a target temperature of 1.1 (in LJ units). The damping factor (0.1) and random seed (933888) control stochastic thermal fluctuations, providing correct equilibrium sampling while the system is under shear.

`fix_modify thermostat temp bias`: Modifies the thermostat to use the `bias` compute, ensuring that the imposed shear flow is not affected by the thermostat. Only the thermal motion of particles is controlled, preserving the physical shear profile.

**Key points:** Shear is generated by moving one wall while keeping the opposite wall fixed. Wall particles are frozen to create rigid boundaries, preventing fluid penetration. Langevin thermostat maintains thermal equilibrium of the fluid without interfering with the imposed flow. Using `compute temp/partial` with `bias` ensures accurate temperature measurement under non-equilibrium conditions. This combination allows measurement of shear stress and velocity profile, which are used to compute viscosity via NEMD.

5. **Measuring temperature under shear**

```
compute bias fluid temp/partial 1 0 1
compute fluid_temp fluid temp/ramp vy 0 $srate x 1 19
```

**Explanation:**

`compute bias fluid temp/partial 1 0 1`: Calculates the thermal temperature of the fluid by removing contributions from the imposed shear flow. Here, the arguments (101) specify that only the $x$ and $z$ velocity components contribute to the temperature calculation, while the flow direction ($y$) is excluded. This ensures that the measured temperature reflects only random thermal motion, not systematic shear-induced streaming.

`compute fluid_temp fluid temp/ramp vy 0 $srate x 1 19`: Evaluates the local temperature profile of the fluid while accounting for the linear velocity gradient imposed by the shearing upper wall. The `temp/ramp` compute subtracts the streaming velocity ($v_y(x)$) from each particle, giving the thermal temperature in bins along the $x$-direction. Arguments `x 1 19` divide the fluid domain into 19 bins along the shear-gradient, providing spatial resolution of thermal fluctuations.

**Key points:**

The thermal temperature under shear is computed by removing the contribution of the flow velocity. This separation is essential for accurate thermostatting and for evaluating viscosity from stress and velocity fluctuations. Binning along the $x$-direction allows monitoring of temperature gradients and ensures the system is thermally equilibrated despite the imposed shear. Using `compute temp/ramp` ensures that non-equilibrium effects (flow) do not bias the measured thermal energy.

6. **Equilibration run**

```
run 100000
```

7. **Production Run and Velocity profile sampling**

```
compute chunk_1 fluid chunk/atom bin/1d x lower 0.05 units reduced
compute chunkvel fluid vcm/chunk chunk_1
fix 1 all ave/time 100 10 1000 c_chunkvel[*] file
velocity_profile.txt
```

**Explanation:**

`compute chunk_1 fluid chunk/atom bin/1d x lower 0.05 units reduced`: Divides the fluid into bins along the shear-gradient direction ($x$) with a bin width of 0.05 in reduced units. The `lower` keyword ensures the binning starts from the lower boundary of the simulation box. This binning is essential for resolving spatial variations in the velocity profile.

`compute chunkvel fluid vcm/chunk chunk_1`: Computes the center-of-mass velocity of the fluid atoms within each bin defined by `chunk_1`. This isolates the collective motion of particles in each layer, producing a coarse-grained velocity profile that reflects the local flow behavior.

`fix 1 all ave/time 100 10 1000 c_chunkvel[*] file velocity_profile.txt`: Averages the center-of-mass velocities over time for each bin to obtain a smooth, steady-state velocity profile $v_y(x)$. The parameters `100 10 1000` indicate: sample every 100 timesteps, average 10 samples at a time, and output every 1000 timesteps.
The file `velocity_profile.txt` stores the averaged velocities for post-processing.

**Key points:**

This setup captures the steady-state velocity profile under shear by averaging both spatially (bins) and temporally (time averaging). Temporal averaging reduces statistical noise and reveals the linear flow profile expected in Newtonian fluids. The output can later be used to extract the shear rate $\dot{\gamma}$ by fitting the velocity profile to a straight line. Proper bin width and averaging parameters are crucial to balance spatial resolution and statistical accuracy.

8. **Shear stress measurement**

```
variable pxy equal pxy
fix pxy_ave all ave/time 100 10 1000 v_pxy file pxy.dat
```

**Explanation:**

`variable pxy equal pxy`: Stores the instantaneous off-diagonal pressure tensor component $P_{xy}$ for each timestep. This quantity represents the microscopic shear stress in the fluid and is central to calculating viscosity in NEMD simulations.

`fix pxy_ave all ave/time 100 10 1000 v_pxy file pxy.dat`: Time-averages the instantaneous $P_{xy}$ values to obtain a smooth measure of shear stress under steady-state shear. Parameters `100 10 1000` indicate: sample every 100 timesteps, average 10 samples at a time, and output every 1000 timesteps. The averaged values are written to `pxy.dat` for post-processing.

**Key points:**

Time-averaging is crucial to reduce thermal fluctuations and reveal the steady shear stress. The resulting averaged $\langle P_{xy} \rangle$ can be directly used in the NEMD constitutive relation

$$\eta = -\frac{\langle P_{xy} \rangle}{\dot{\gamma}}$$

to compute the shear viscosity. Correct sampling frequency and averaging interval ensure accurate statistics while capturing steady-state behavior. The variable and fix setup separates the instantaneous microscopic stress from the time-averaged macroscopic stress, facilitating clear analysis.

### 3.5.2 Instructions

To get the viscosity by using the created LAMMPS script, the following steps can be followed.

1. Run the LAMMPS input file.
2. Run the given python file to calculate viscosity, which will print the same and export and save the relevant plots.

The Python script `viscosity_analysis.py` performs:

1. Loads the time-averaged velocity profile and computes mean velocity in each bin.
2. Excludes bins adjacent to walls to remove boundary-layer effects.
3. Fits a straight line $v_y(x) = \dot{\gamma} x + c$ to the steady-state profile to extract shear rate $\dot{\gamma}$.
4. Computes time-averaged shear stress $\langle P_{xy} \rangle$.
5. Evaluates viscosity using $\eta = -\langle P_{xy} \rangle / \dot{\gamma}$, again obtained from the LAMMPS exported files, specified during the LAMMPS input file setup.
6. Produces diagnostic plots for velocity profile, shear rate, and stress time series.

Note: Outcome of this experiment can be dependent on the rate of shear. However the best result is the one which shares the value commonly for the different shear rate. Also, getting smooth, linear and plane velocity gradient is very crucial for the correct calculation of viscosity.

### 3.5.3 Learning Outcome

After completing this exercise, students will be able to:

1. Understand how nonequilibrium molecular dynamics (NEMD) can be used to compute transport properties such as viscosity.
2. Relate the imposed shear rate to the velocity gradient measured in the system and identify the steady-state linear profile.
3. Analyze the microscopic origin of shear stress by examining particle momentum transfer under flow.
4. Calculate the shear viscosity $\eta$ using the relation $\eta = -\langle P_{xy} \rangle / \dot{\gamma}$, connecting macroscopic transport coefficients to microscopic quantities.
5. Recognize and exclude boundary-layer effects to obtain accurate bulk viscosity measurements.
6. Interpret diagnostic plots of velocity profiles, shear rate, and stress fluctuations to evaluate system equilibration and data reliability.
7. Develop familiarity with combining LAMMPS simulation output with Python analysis for post-processing and visualization.

## 3.6 Langevin Dynamics and Fokker-Planck Equation

Use the folder: `Hands_on/langevin_dynamics_vs_fokker_plank_equation`
LAMMPS input file: `langevin.in`

This exercise illustrates the equivalence between stochastic particle-based dynamics (Langevin equation) and the continuum description of diffusion provided by the Fokker–Planck equation. Using LAMMPS, a large ensemble of non-interacting particles is evolved under Langevin dynamics and the resulting probability density evolution is compared with the analytical solution of the Fokker–Planck equation.
The LAMMPS input file used for this simulation is `langevin.in`.

### 3.6.1 System Setup

The following steps describe how the system is constructed and evolved in the LAMMPS input script.

1. **Definition of a large simulation domain and localized initial condition**

```
region box block 0 20 0 20 0 20
create_box 1 box
region region block 10 11 10 11 10 11
create_atoms 1 random 1000 8009 region
```

   A large cubic simulation box is first created. Particles are then initialized inside a small subregion near the center of the box. This configuration represents a sharply localized initial density distribution, closely resembling a delta-function initial condition at time $t = 0$.

   Such an initial condition is ideal for studying diffusion, as the subsequent spreading of particles can be directly compared with the analytical solution of the Fokker–Planck equation.

2. **Removal of inter-particle interactions**

```
pair_style none
```

   Inter-particle forces are completely disabled so that particles behave as ideal, non-interacting Brownian walkers. This ensures purely diffusive motion governed only by thermal noise and friction.

   If interactions were present, short-time correlations and caging effects could lead to deviations from simple diffusion, complicating the comparison with the Fokker–Planck description.

3. **Application of Langevin thermostat and time integration**

```
fix thermostat all langevin 1.0 1.0 0.1 89080
fix integrator all nve
```

The Langevin thermostat introduces both frictional and stochastic forces, modeling the effect of an implicit heat bath. This stochastic equation of motion corresponds, at the ensemble level, to the Fokker–Planck equation governing the evolution of the probability density.

Since the Langevin thermostat does not integrate particle trajectories by itself, a separate time integrator is required. The `nve` integrator is used here, as temperature control is already handled by the stochastic thermostat.

Using an additional thermostat-based integrator (e.g., `nvt`) would result in double thermostatting and is therefore avoided.

### 3.6.2 Instructions

To analyze the equivalence between Langevin dynamics and the Fokker–Planck equation, follow the steps below:

1. Run the LAMMPS input script `langevin.in` to generate the stochastic particle trajectories.
2. Open OVITO and load the trajectory file `simulation_data.LAMMPStrj`.
3. Apply the *Histogram* modifier in OVITO and monitor the evolution of the particle number density along the $x$-direction as a function of time.
4. Observe the gradual spreading of the initially localized particle distribution, characteristic of diffusive motion.
5. Compare the OVITO histogram with the analytical solution of the Fokker–Planck equation shown in the animated file `fokker_planck_evolution.gif`, which is generated using the Python script `evolution_focker_planck.py`.
6. Verify that the stochastic particle-based Langevin simulation reproduces the same macroscopic density evolution predicted by the continuum Fokker–Planck equation.

Note: This exercise highlights the fundamental connection between microscopic stochastic equations of motion and continuum diffusion equations. The thermostat are used explicitly. However, the same diffusive behavior is expected if we remove the thermostat and just apply the simply apply the `nve` integrator.

### 3.6.3 Learning Outcome

After completing this exercise, you are supposed to:

1. Understand how Langevin dynamics introduces stochastic and dissipative forces to model thermal fluctuations at the particle level.
2. Learn how to set up and run particle-based simulations that represent diffusive motion in the absence of inter-particle interactions.
3. Comprehend the role of an initially localized particle distribution as a delta-function-like initial condition for diffusion.
4. Apply spatial binning and histogram analysis to extract time-dependent particle density profiles from simulation trajectories.
5. Observe and interpret the diffusive spreading of particle distributions as a function of time.
6. Recognize the correspondence between stochastic particle trajectories and the macroscopic evolution described by the Fokker–Planck equation.
7. Compare discrete simulation results with analytical continuum solutions and verify their quantitative agreement.
8. Develop intuition for how macroscopic transport equations emerge from microscopic stochastic dynamics.
9. Gain practical experience in using visualization tools and post-processing techniques to validate theoretical predictions using molecular simulations.

# References


[1] **Computer Simulation of Liquids** by Allen and Tildesley.

[2] **Theory of Simple liquids** by Hansen and McDonald.

[3] **Nonequilibrium Statistical Mechanics** by Robert Zwanzig