Manual for running the BD Codes

February 6, 2025

1 Instructions for running the codes stored in folder: t2_langevin_focker_planck

This section describes the evolution of particle distribution over time, where the particles follow Langevin stochastic dynamics. The corresponding LAMMPS input file is named `t2_langevin.in`.

Configuring the System

To configure the system, system-specific changes are made in the script. For details, please refer to Sec. 1 of the DPD LAMMPS manual.

1. **Creating the particles in a narrow region within a large simulation box:**

   ```
   region my_box block 0 20 0 20 0 20
   create_box 1 my_box

   region my_region block 10 11 10 11 10 11

   create_atoms 1 random 1000 8009 my_region
   ```

   A large simulation box is created, and particles are generated in a narrow region within the simulation box. In practice, this results in a delta distribution of particles at time $t = 0$.

2. **Switching off the pair coefficient:**

   ```
   pair_style none
   ```

   The pair style is switched off to simulate ideal diffusing particles at all length scales, ensuring smooth evolution. If not switched off, particles will experience caging, leading to subdiffusive behavior at smaller time scales.

3. **Implying an integrator and a thermostat:**

   ```
   fix my_thermo_stat all langevin 1.0 (#start T) 1.0 (#end T) 0.1 89080

   fix your_integrator all nve
   ```

The Langevin thermostat is applied using the `fix` command, which effectively adds friction terms to maintain constant temperature by controlling the divergence of velocity. However, the thermostat alone does not integrate the motion of particles. Therefore, a separate integrator, such as the simplest one `nve`, is used in this example.

If particles have finite sizes (e.g., not point particles), an integrator capable of handling rotational degrees of freedom, such as `rigid/small molecule`, would be necessary. Since the system is already thermostatted using Langevin, using `nvt` for integration would introduce redundant thermostats (e.g., Nose-Hoover or Anderson), potentially leading to conflicts.

In conclusion, the choice of integrator and thermostat should be compatible with each other and the system being simulated.

Steps to Follow:

To perform the scaling of time, students can follow the steps below:

1. Run the code stored in the folder named `t2_langevin.in`.
2. Open OVITO and load the data file `simulation_data.lammpstrj`.
3. Open the Histogram filter and observe the evolution of the system's position along the x-axis.
4. Compare the system's evolution observed in LAMMPS with the results of the Fokker-Planck equation stored in the GIF file named `fokker_planck_evolution.gif`. This GIF is generated using the Python code `t2_evolution_focker_planck.py`.
5. Observe that the Langevin dynamics, which models discrete individual particle behavior, reproduces the same overall system evolution as predicted by the continuum Fokker-Planck equation, demonstrating consistency between the two approaches.

2 Instructions for running the codes stored in folder: t2_patchy_particles

Now you are going to learn how to create a Brownian particle and decorate it with attractive patches for directional bonding. Additionally, you will thermostat both the translational and rotational degrees of freedom using the Langevin thermostat to perform a Brownian dynamics simulation.

First, please read the explanation below regarding the system configuration, which highlights specific configurations in the LAMMPS script file t2_patchy_particles.in.

Configuring system:

To simulate a Brownian system, make the following changes in your LAMMPS script:

Note: For more details, please check Sec. 1 of the DPD manual.

1. **Defining particle's attributes:**

```
units lj
atom_style  hybrid sphere molecular
boundary p p p
pair_style hybrid lj/cut 2.0 cosine/squared 0.12
```

   Your Brownian particle is a solid sphere, which also has angular momentum. The atom style hybrid sphere molecular enables the particle to possess attributes like angular momentum and allows a rigid shape if merged with patches using the molecule part of the atom style, which are also particles.

2. **Making memory for two types of patches:**

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

   The command create_box 2 box ensures memory allocation for two kinds of particles. One will be the heavy core particle at the center, and the other type will be the patches.

3. **Loading molecule from a template:**

```
molecule patchy_part t2_patchy_molecule.mol
create_atoms 0 random 5000 87910 NULL mol patchy_part
... 454756 overlap 1.5 maxtry 50
```

   A molecule can be imported from a molecule template file, which contains information like type ID, position, particle ID, bonding information, etc. When creating atoms from the template, assign atom type ID 0, as it is already supplied by the template file. The template file t2_patchy_molecule.mol allows modifying particle positions to adjust patch configuration, shape, size, etc.

   Avoid overlapping hard cores using the command overlap 1.5 maxtry 50. If LAMMPS cannot find free space for a particle within 50 trials, it will not generate that particle. This method is unsuitable for generating high-density systems. Instead, create a low-density system with a large box size, compress it using an NPT ensemble at a given pressure, and then switch to an NVE ensemble after unfixing NPT ensemble. However, in the case being studied, low volume fraction (or density) is sufficient.

The molecule template stored in the file `t2_patchy_molecule.mol` is shown below:

```
#number of atoms
5 atoms

Coords
#particle id coordinate
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
# particle id  particle type id
1   1
2   2
3   2
4   2
5   2
```

4. **Defining interaction between the core particles and the patches:**

```
pair_style hybrid lj/cut 2.0 cosine/squared 0.12
pair_coeff 1 1 lj/cut 0.01 1.5 2.0
pair_coeff 1 2 none
pair_coeff 2 2 cosine/squared 8 0.3 0.35 #square well attractive sites
```

We are defining the hard-core spherical particle using Lennard-Jones (LJ) interaction. The patches are made attractive to each other, but the patches do not interact with the cores. Effectively, we are creating a molecule with a spherical core and attractive sites (patches) such that whenever these sites come close to each other, they attract.

5. **Defining the physical properties like mass and the diameter of the particles:**

```
set type 1 mass 1.0
set  type 2 mass 0.000001

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

The patchy particle is treated as a point particle with almost negligible mass and size, as it is not an actual particle but just an attractive interaction site. The interaction has already been considered while defining the pair style (the command `pair_coeff 2 2 cosine/squared 8 0.3 0.35`).

6. **Grouping particles:**

```
group core type 1
group patch type 2
group rigid_molecule type 1 2
```

   Fixes in LAMMPS are either applied to all particles using the keyword `all` or to specific groups. Fixes cannot be applied directly to atoms by their ID. Therefore, atoms are grouped into sets, and fixes are applied to these groups by referencing their group names.

7. **Commands to speed up simulation by switching off unnecessary calculations:**

```
neigh_modify exclude molecule/intra rigid_molecule ...
...every 1 delay 0 check no
```

   This command ignores the intra-particle interaction calculations, e.g., among the patches belonging to the same molecule. These calculations are unnecessary as particles within the same body behave like a rigid body.

8. **Fixing the thermostat and applying the integrator:**

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

   The Langevin thermostat is applied only to the cores. There is no need to thermostat the patches, as they are not actual particles but just massless interaction sites. The command `omega yes` thermostats the rotational degrees of freedom. When using the integrator `rigid/small molecule`, it treats the entire molecule as a single rigid body. Forces exerted on the patches contribute to the overall force on the molecule, which is accurate since the patches induce interactions that influence the motion of the cores.

9. **Precautions while measuring thermodynamic quantities:**

```
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
```

   When LAMMPS computes temperature, it considers all particles, including the patches, which are not actual particles due to their negligible mass and radius. This leads to incorrect temperature values as patches are not thermostated. To address this, the calculations for thermodynamic quantities such as kinetic energy and temperature are applied only to the core group of particles. This ensures an accurate representation of the system's thermodynamic state.

   Note: For more details about the `compute` command, refer to **Sec. 1 of the DPD manual.**

Steps to follow:

Please follow the following steps to create and simulate a system of patchy particles,

1. Customize the molecule template according to your requirements. Ensure that the first row of the template file is either commented or left blank; otherwise, LAMMPS will show an error.

2. Run the script file `t2_patchy_particles`.

3. Load the output data in OVITO and visualize the self-assembly of the patches.

4. Lower the temperature by using the command:

   ```
   fix thermo_stat  core langevin 1.0 1.0 0.1 428984  omega yes
   ```

   Refer to the LAMMPS manual for details about this fix and how to set the temperature. Typically, you set the start and end temperatures to the same value, e.g., `1.0 1.0`, as shown in the example. The damping parameter (`0.1` in this case) is chosen to control the friction in the system:

   - Lowering this value increases damping, reducing diffusivity by increasing the friction in the system.
   - Increasing this value reduces damping, increasing diffusivity, and lowering the friction in the system.

5. Observe the clustering of Brownian molecules at low temperatures.

3 Instructions for Running the Codes Stored in Folder: t2_liquid_gas_coexistence

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. And finally, calculate the density profile of the combined system, after equillibrating the system.

To achieve this, you will use multiple LAMMPS input files, each configured as described below.

System configuration:

To make a high density liquid regime, you will use the file named t2_liquid_box_creator.in. The description of a few key commands are given as below:

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

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

   All the details have already been explained.

2. **Applying NpT ensemble to compress the fluid and make it a liquid phase:**

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

   After generating the box throughout the whole box, you use a barostat along the z-axis with some pressure. We use a guessed pressure where the system can exist in the liquid phase and apply that pressure. For example, in the given script, you are applying the temperature `0.4 0.4`, and the pressure along the z-axis as `10 10`. Applying pressure along only the z-axis does not mean that in other directions the pressure will be different. It is just for the shape of the box, where the box will be compressed along the z-axis to bring the system to the specified pressure.

3. **Printing the last configuration after the equilibration:**

```
write_data compressed_coordinate.data
```

   We use the command to print the last configuration. The NpT thermostat and barostat have compressed the box along the z-axis such that the box will now be located in the middle of the initial box. For example, if the box initially extended between $0 - 90$, then after compression, it might be located between $38 - 52$.

After generating the liquid phase, the next step is to merge it with the gas phase. To achieve this, we take the previously generated and equilibrated box with the extension from 38 to 52 along the z-axis and place it into a larger box that spans from 20 to 70. The empty regions, from 20 to 38 and from 52 to 70, are then filled with a low-density gas phase. This is done by randomly placing an appropriate number of gas particles.

If the number of gas particles is too low, it will result in a lower chemical potential, causing some of the liquid-phase particles to evaporate and transition into the gas phase. Conversely, if there are too many gas particles, the chemical potential of the gas will be higher, prompting some of the gas particles to condense and deposit into the liquid phase. This process will continue until the vapor pressure and chemical potential reach their equilibrium values at the specified temperature.

To merge the two phases, we place the liquid phase into a larger box that extends further along the z-axis. The box is enlarged by editing the particle data, stored in the file named `compressed_coordinate.data`, which was output by the script `t2_liquid_box_creator.in`. However, it is important to remove the periodic neighborhood along the z-axis, as the particles that were previously counted within the interaction range are no longer periodically connected. This is due to the wall shifting away from the particles at both ends (e.g., at lengths 38 and 52).

To remove the periodicity along the z-axis, we use a separate script called `t2_intermediate.in`, which loads the particle's data and rewrites with the same file name, after running for 0 time steps. After removing periodicity, you can change the box size by altering the last generated particle data file `compressed_coordinate.data`.

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

```
boundary p p s
```

   In this script, the periodic conditions are modified only for the z-axis. The `s` indicates that the particles are no longer connected through the periodic boundary in the nearest-neighbor list along the z-axis. This ensures that the wall applied to the system fully encloses all the particles along the z-axis, leaving no particles excluded.

   After removing the periodicity, the updated data is written to a new file, which is then loaded by the script named `t2_coexistence_simulator.in`. In this script, the system data is loaded, and regions are defined on either side of the liquid phase, where particles will be added.

2. **Defining the 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. maxtry 50
create_atoms 1 random 100 87910 gas_region_2 overlap 1. maxtry 50
```

   In this step, two regions are defined, but both are within the overall simulation box. These regions are then filled with gas particles.

3. **Binning the number of particles 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
```

   This command is used to print the number of particles in each spatial segment along the z-axis. The z-axis is divided into bins of size 0.02 (in reduced units), and the number of particles in each bin is counted. For further details, consult the LAMMPS documentation or feel free to ask for additional clarification.

Note: The rest of the description of the code remains unchanged.

Steps to follow:

To solve the relevant question, students are advised to follow these steps:

1. Run the LAMMPS script `t2_liquid_box_creator.in` after setting the desired value of pressure, which ensures the high-density liquid phase. If you do not create a high-density liquid phase, during the simulation with the gas phase, the gas bubbles (which act as nuclei of the gas phase) will grow, leading to a fragmented region of liquid and gas.

2. Run the script `t2_intermediate.in`, which loads the file `compressed_coordinate.data`.

3. Now you have the file `compressed_coordinate.data`, with periodic boundary conditions switched off along the z-axis. Open the data file using a text editor and change the box length, extending it symmetrically on both sides of the z-axis. For example, if the box length along the z-axis is `38 52 zlo zhi`, extend it to `20 70 zlo zhi`.

4. Save and close the file `compressed_coordinate.data`.

5. Run the script `t2_coexistence_simulator.in`, which will equilibrate the system and generate the number of particles in each bin along the z-axis at different time steps. The data will be stored in the file named `bin_particles.lammpstrj`. This data will be used later to calculate the density distribution.

6. Run the Python script named `t2_data_rectifier.py`, which will load the binned data for the number of particles and average it over time. The script will then print the density distribution ρ versus z in a text file named `averaged_density_profile.txt`.

7. Open the DESMOS online graph plotter in your browser and type the function $tanh(x)$. Observe the shape of the curve. Near the gas-liquid interface, you will notice a similar density distribution.

8. Run the Python script `t2_density_profile_fitter.py`, which will fit the ρ versus z data to a *tanh* function. The terminal will display the gas and liquid coexistence densities. A plot of the simulated data and the curve fit will also be exported as a PNG file named `density_profile_fit.png`.

4 Instructions for running the codes stored in folder: e2_brownian_dynamics

In this folder, you have codes that generate the Mean Squared Displacement (MSD) for the particles to calculate the diffusivity.

Below are the specifications for the script `e2_brownian_dynamics.in`.

System Configuration:

For details, refer to the previous sections.

The code generates particles with a homogeneous distribution along the x, y, and z axes. Particle interactions are switched off (students should try to identify the relevant section, based on the instructions from previous parts). The particles diffuse according to Langevin dynamics, and the script outputs the MSD data.

1. **Setting up the initial temperature and velocity distribution:**

```
# Assigning initial velocity distribution
velocity all create 4.0 12345 mom yes rot no
```

2. **Setting up the temperature in the Langevin thermostat:**

```
# Assigning initial velocity distribution
fix my_thermo_stat all langevin 4  4 .1 89080
```

In this section, we define the thermodynamic parameters such as temperature (given as `4.0` in the example script) and the damping parameter (given as `0.1`), which is the inverse of the friction coefficient.

Caution: Please ensure that the temperature in the initial velocity distribution command is the same as the one used in the thermostat.

Steps to Follow:

1. Open the file named `e2_brownian_dynamics.in` and set the desired temperature.
2. Run the script and wait for the output data.
3. Run the Python script `e2_msd_data_rectifier_and_plotter.py`, which will plot the MSD versus time data and provide the value of the diffusivity.
4. Change the mass of the particle and record the diffusivity, while keeping all other parameters the same. Then plot mass versus diffusivity, saving the data in a text file **.txt**, and plot it using Python (or use your preferred plotting method).
5. Change the friction coefficient of the system while keeping all other parameters the same, and plot friction versus diffusivity.
6. Change the temperature while keeping all other parameters the same, and plot temperature versus diffusivity.

5 Instructions for running the codes stored in the folder: e2_binary_mixture

This system creates a binary mixture of particles using the script e2_binary_mixture.in. For the details of the commands, students are referred to the previous sections. However, a few key configurations are outlined below.

System Configuration:

The code generates a binary mixture with particles randomly and homogeneously distributed across the entire box. By adjusting the effective radius of the Lennard-Jones (lj) particles, the mixture can be made either additive or non-additive. An additive mixture is defined as $d_{ij} = \frac{1}{2}(d_{ii} + d_{jj})$. Any other value for d_{ij} will be considered a non-additive mixture.

1. **Setting the additive and non-additive binary mixture:**

```
pair_style lj/cut 2.4
pair_coeff  1  1  0.01  1.   1.5
pair_coeff  1  2  0.01  1.55  2.1
pair_coeff  2  2  0.01  2.   2.2
```

   In the interparticle interaction description, you are using the Lennard-Jones potential. The σ value determines the size of the hard core. For the particle types with ids 1 and 2, the example script sets $\sigma = 1.55$, which is slightly different from the additive condition $d_{ii}^{\text{add}} = \frac{1}{2} \cdot (1 + 2) = 1.5$.

2. **Setting the appropriate pressure to observe phase separation:**

```
fix my_pressure all npt temp 1 1 1 iso 20 20 1.0
```

   In this script, you are using the *npt* ensemble with a defined temperature and pressure value, for example, iso 20 20 1.0, which fixes the pressure at 20.0. Setting the pressure is necessary because the phase separation observed in non-additive mixtures is purely entropic in nature, unlike DPD, where interactions are considered with different strengths by using varying force constants. As the density increases, the system, which aims to maximize entropy, will try to find more phase space for the particles to move. In a non-additive mixture, more phase space becomes available when the particles are separated from each other.

3. **Removing the NpT ensemble:**

```
unfix my_pressure
```

   Once the system has equilibrated at a given pressure, the NpT ensemble is removed, and the dynamics are applied to the system, allowing it to evolve and equilibrate further.

Steps to Follow:

1. Set the pair coefficients σ and pressure in the script e2_binary_mixture.in.

2. Run the code.

3. Load the simulation data file `simulation_data.lammpstrj` in OVITO.
4. Observe the phase separation, where you will notice clusters of particular types of particles seperated from the other type of particles.
5. Change the pressure for the same inter-particle pair coefficient σ, making it high (around $10 - 15$) and low (around $0.05 - 1$).
6. Observe the effect of pressure on phase separation for that particular temperature and pair coefficient.

6 Instructions for running the codes stored in the folder: e2_patchy_particles

This folder contains a patchy particle template and the LAMMPS script to run the simulation. Students are already familiar with all the commands and functions used in the input script e2_patchy_particle.in. However, a few aspects are clarified here.

System Configuration:

1. **Setting the molecular template:**

```
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
```

The template sets up two types of particles: type id 1 for the central core particle and type id 2 for the patches. The positions of the patches are arranged around the central core particle, which has type id 1. You can modify the number of patches by changing the atom count (`5 atoms`) and adjusting the `types` and `Coords` accordingly.

Steps to Follow:

1. Customize the molecule template provided in the file e2_patchy_molecule.mol.
2. Run the script e2_patchy_particle.in and load the output simulation data into OVITO.
3. Verify that the patches are located around the core, with the desired geometric configuration.

Appendix

Using OVITO for Visualization

1. Load the trajectory file: `File > Load File`.
2. Apply `Modifiers` to enhance visualization (e.g., `Color Coding`).
3. Adjust the perspective for better screenshots.

Using `matplotlib` for Plotting

1. Install `matplotlib` using `pip install matplotlib` if not already installed.
2. Run the provided plotting script, replacing placeholders with actual data.
3. Save the generated plot as a PNG file.
