Manual for running the DPD Codes

January 1, 2025

1 Instructions for running the codes stored in folder: t1_dpd_time_scaling

This section describe the scaling of time in dpd simulation, where you use the LAMMPS input file named t1_time_scaling_dpd.in.

Configuring system:

1. **Defining variables:**

```
# Some variables
variable T    equal 1.0
variable rc   equal 1.0
variable rcD  equal 1.0
variable  L   equal 10
```

LAMMPS allow to define many kinds of variables. Some remains constant through out the simulation and defined globally, while others changes at every time steps whenever accessed to compute the movement of any atom. So for each atom and each time step the variables gets recomputed. However the example variable in this example remains constant throughout the simulation and also for each particles.

2. **Essential simulation settings:**

```
# Basic simulation settings
units        lj
boundary     p p p
atom_style   atomic
dimension    3
```

These variables are essentials and need to be defined for every LAMMPS simulation. Units lj sets all the mass, length, energy, temperature and derived time unit in terms of the potential specified. For example if lj potential have potential well depth 3 unit then the temperature k_BT will also be in the same unit. So if temperature is 4 unit then, its 1 unit greater than the the potential well depth defined in the simulation. In DPD you don't use lj potential, so the temperature if you will keep 1, then temperature unit

becomes equal to potential width unit. Now, all the other variables values can be guessed directly, as it will be implemented in the temperature unit (k_BT).

Boundary condition `p p p` ensures the periodic boundary condition. It can also be switch off and can be made non periodic expanding boundary, where boundary will shift with the atom diffusing away from the box etc. There is also a non periodic non expanding boundary where the atoms get lost once crossing the boundary (please see the LAMMPS manual).

Atom style `atomic` consider a point particle. There is no atom electron and nucleus being considered so please be aware. For example atom style molecule will allow applying bonding between the particles. Again there is no any quantum bonding but just classical harmonic spring (or any other kind of classical force), defined to make a bond. There is no electron cloud or quantum mechanical calculation involved in these commands. For atom style sphere, you can have angular momentum defined and applied, which is not possible for the point particles defined by atom style atomic. In next tutorial when using a Brownian large particle you will use atom style sphere (or hybrid along with molecular, to account both angular momentum and bonding configuration).

In LAMMPS, you can define either dimension 2 or 3. In all cases you are going to simulate in dimension `3`.

3. **DPD specific variables:**

```
newton       on
comm_modify  vel yes
```

These commands are specific for the DPD simulation. If you will keep newton on, then during the simulation the pairwise interaction will be stored and used by the DPD. As you have been taught in your classes that the DPD uses all the forces in pair with the surrounding particles. Even to compute friction force it will find the nearest neighbor within the potential range and use the pair velocity to calculate the friction force. Therefore along with the position of the nearest neighbor, which happens by default in LAMMPS, you will also need to communicate the velocities, which is done by the command comm_modify vel yes, which communicates the velocities of the neighboring particles.

4. **Creating a simulation box:**

```
# Create simulation box
region  your_Box block 0 ${L} 0 ${L} 0 ${L}
create_box  1 your_Box
```

Off course to simulate particles, you will need something, where you can put your particles, a box, where particle will move. To keep the list of nearest neighbor, LAMMPS simulation makes grid, each grid is allocated to some memory where you can store the id of the particles. The memory is limited, therefore you have grid expansion limited. In other words, you will define a box with some extension. And to get rid of simulation artifacts due to the space restriction, you remove the finiteness of box. This is the reason that, you have used the periodic boundary condition in the previous section.

LAMMPS creates regions, which can be rectangular (using the **block** keyword), cylindrical (using the c**cylinder** keyword) etc.

After you define a region you can create a simulation box, which will expand the defined region.

The command **create_box** is important in the sense that the you will have to allocate all the memories here. So the degree of freedom of particles is implemented in the code (whether atom or molecule or sphere) using atom style. But the memory allocation happens here. So if you want to create a binary system, where you have two types of particles, you will use the command `create_box 2 your_Box` command instead of `create_box 1 your_Box`. Similarly, you will specify, number of bonds per particles and then list of bonded neighbors, list of dihedral bonds (all per particle) etc.

Note: If you get any error like list index out of range, or any such error, where it indicates about the shortage of memory, then please try to look at this command (`create_box`) in LAMMPS manual.

5. **Creating atoms in a defined region:**

```
# Create_atoms 1 in region simBox
create_atoms 1 random 3000 12456 your_Box
```

This command creates the atoms within the given region. All the particles must be generated within the box. For example if you define another region, which is larger than the region used for creating the box and try to generate the atoms in that region, you will get the particle generated only in the area which intersects the box extension. All the particles with their randomly assigned positions being beyond box extension, will not be generated.

6. **Defining the particle's properties:**

```
mass         1 1.0
```

This command define the particle's mass, which will be used to calculate the acceleration, velocity and then position by using equation of motion. There are different properties also like radius of particles. But atom style **`atomic`** will not allow those attributes to implemented, as it is just the point particle and implements only the position. For finite size particles, you can use atom style **`sphere`**.

7. **Defining the interaction among the particles:**

```
# Define the pairwise interaction style (DPD)
pair_style       dpd ${T} ${rcD} 3854262
pair_coeff   1   1  78.0  4.5  1.0
```

Now it comes about defining the inter-particle interaction. You can also switch off the inter-particle interaction by using command **`pair_style none`** (no need to define pair coefficients). In that case, your system will become a hot Lorentz gas with its pressure and energy defined by the ideal gas equation of state. **`pair_style lj cut/off 2.5`**, is the most widely used pair style (where 2.5 is the cutoff length).

All the pair styles have their own unique way of defining coefficients, used in that interaction. But in all that case, once you define pair style and then pair coefficient. In defining pair coefficient, you also tells that between which kind of particles, what the coefficient will be.

In your case, you are using DPD pair style.

For the detailed description and to find the meaning of these numbers, you can check LAMMPS manual. These will also be discussed in detail in the next section.

8. **Defining the integration style, simulation time step and initial velocities:**

```
# Defining some simulation parameters
run_style    verlet
velocity all create ${T} 68768932
timestep 0.005
```

If you integrate equation of motions to calculate the velocity and the position in the next time step. Then you will find a dilemma about choosing the value of acceleration and velocity defined at beginning; t, and end of the time step; $t + dt$. Because of the chaos system can diverge in the large time scale just because of minute error in forces and velocity. There is whole formalism developed to define what acceleration and velocity values must be chosen so that the system converge to the real and accurate configuration. Such an scheme is called Verlet algorithm, which is defined by using command **`run_style verlet`**. If you don't define, its fine, LAMMPS will take some default integration style.

Before simulation you can define a random velocity chosen from the Maxwell Boltzmann distribution. In this way, your system equilibrate faster. However defining equilibrium velocity distribution for a particular temperature, which is nothing but MB distribution, it does not guarantee that your system will kick start from a thermodynamic equilibrium. As you have also the position dependent potential energy implemented by pair interaction style. Therefore, it takes time to equilibrate the system. After equilibration, each degree of freedom will have stored the same energy and equal to $k_B T/2$ (following the law of equipartition theorem).

Choosing small time step will take more simulation time to achieve the same physical time. And choosing larger time step will can cause the overlap of the infinite core creating high infinite acceleration and the particle will run off the box and simulation will show the error. Therefore you can observe a calibration between accuracy and efficiency. So please chose the time step wisely it should be sufficiently small but not too small to cost computational power.

9. **Showing the parameters on terminal:**

```
# Output thermodynamic quantities which shows on terminal
thermo_style custom step time temp press
thermo       100
```

This command is defined to show some quantities on terminal. LAMMPS calculates some default quantities without any need to evoke any compute command. Such as time, temperature, pressure, kinetic energy, etc.

The command **`thermo 100`** prints, at each hundredth simulation time steps.

10. **Apply an integrator:**

```
# Apply the NVE ensemble
fix your_fix all nve
```

An integrator is applied by using fixes. So there are specific commands called fixes. Fixes can be applied to a selected group of atoms, which will be discussed in the later section. If you don't use any specific group, then apply it to all kinds of atoms, using **all** value. You can unfix any fix and then rerun the same system. For example you can compress a box using *Npt* fix and then unfix and apply *nvt* ensemble.

Not all fixes are integrators. And to move the particles, and update velocity and position, you need an integrator like **nve, nvt, npt, rigid/small molecule**, etc. Without an integrator, which is applied in the system using fix command, you will not get your simulation running and nothing will move with the increase in the simulation steps.

11. **Run the simulation:**

```
# Equilibration phase - run to allow the system to stabilize
run          200000
```

Now you run your code by defining the simulation steps. The physical time will increase in the multiple of *dt* (which you defined by using**timestep** command).

You often first run the simulation just to equilibrate the system. Later run will be performed, after applying the command to save the data. As done below.

12. **Using built in computational fascility of LAMMPS:**

```
# Compute MSD (Mean Squared Displacement) after equilibration
reset_timestep  0
compute your_msd_all all msd
fix your_msd_output all ave/time 100 1 100 c_your_msd_all...
...file your_msd.dat mode vector
```

After running for the equilibrium, you can reset the time as 0. It will make your computation easy.

Before running for production, you can also use some builtin **compute** command to compute some quantities. There are a plenty of compute commands builtin in LAMMPS to compute various dynamical physical quantities like pressure tensor, kinetic energy, special histogram of particles and many more. See the LAMMPS manual for the detail.

Once used compute command, averaging and printing the data will be required. Which is done by the command **fix your_msd_output all ave/time 100 1 100 c_your_msd_all file your_msd.dat mode vector**. Please see the LAMMPS for the detail of **ave/time**.

For example if you are defining a compute variable **your_msd_all**, you will call that compute variable by using the name **c_your_msd_all**. Remember different kind of variables are called in a specific way, which you can learn from the LAMMPS manual.

13. **Running again to collect the data after ensuring the system equilibrium:**

```
# Production run to measure MSD over time
run          200000
```

Once you define all the quantities which you need to print within the simulation, using fixes, you run for the production.

Steps to follow:

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

1. Run the code stored in the folder named `t1_time_scaling_dpd.in`.
2. Run the python code `t1_msd_data_rectifier_and_plotter.py`. The code will analyze the output MSD data and print the diffusivity. It will also export a plot with relevant information.
3. You have MSD in DPD length unit for the DPD beads. Use the relation to convert the MSD in real unit:

$$t_{\text{phy}} = \frac{N_m \cdot (30.0\rho N_m)^{2/3} \langle R^2_{\text{cm}} \rangle}{D_{\text{water}}(\text{in unit of } \hat{A}^2/s)}$$

Where R^2_{cm} is your MSD for the DPD beads which got from your python code. The factor N_m changes the center of mass bead displacement to the displacement of particles constituted by the beads. The factor $\cdot(30.0\rho N_m)^{2/3}$ convert the total displacement of the DPD particles (which is $N_m R^2_{\text{cm}}$), to the real time unit in $\hat{A}$. D_{water} is already given in the real unit for the actual water molecule.

Length is scaled by volume of ρN_m (number of DPD molecule in the one unit volume, in DPD length) real molecule. Where $30\hat{A}^3$ is the volume of single molecule in the real system. So one unit volume is equivalent to $30 \cdot \rho N_m \hat{A}^3$ in real unit. So one length unit in DPD will be equal to $(30 \cdot \rho N_m)^{1/3}\hat{A}$.

4. Once you compute the physical time in real unit covered in the given simulation physical time. You can find out the relation between the simulation physical time and the real physical time using the expression.

$$t_{\text{f}} = \frac{t_{\text{phy}}}{t_{\text{sim}}}$$

5. Once you have the t_f, you can convert any simulation time to the real time unit just multiplying that t_{sim} with the factor t_f

Note: Students must not be confused with the simulation steps as the simulation time t_{sim}. As t_{sim} is given by $t_{sim} = simulation\_steps \cdot dt$

2 Instructions for running the codes stored in folder: e1_rho_vs_pressure

Since you can define your time scale for any force constant A and density ρ, so that you get the same diffusivity from DPD, after unit conversion, and equal to the diffusivity value in the real system. Then what will be the appropriate choice of force constant and densities? The related questions asked in your exercise exactly points at this aspects.

By solving this exercise you will come to know about the appropriate ρ value. In the next exercise you will come to know about the appropriate force constant A (or a_{ij}).

So in practice, you first fix density ρ and then force constant a_{ij} with an appropriate value of σ (strength of noise). While the cutoff is usually set to 1 for simplicity. And than calibrate your length and time scale accordingly, from diffusion coefficient calculated from the simulation, as you did in your tutorial.

But first please read the below explanation of the system configuration, which focuses more on the pair coefficient defined in your LAMMPS script file `e1_rho_and_pressure.in`.

Configuring system:

To simulate DPD in your LAMMPS package you use the following commands:

1. **Defining DPD pair coefficient in LAMMPS:**

```
# Define the pairwise interaction style (DPD)
pair_style       dpd ${T} ${rcD} 3854262
pair_coeff   1   1  25.0  4.5  1.0
```

It is easy to guess how the LAMMPS implement values in the code if you will keep T as 1. As in lj unit energy and temperature is defined in ϵ unit, where ϵ is the potential depth of the lj potential. If you define k_BT (which we call T) as 1. Then you can guess what will be the other quantities like force constants etc. For the full detail please see the LAMMPS manual. Always keep **`rcD`** greater than the cutoff defined in the **`pair_coeff`** section.

You have defined **`pair_coeff`** between particle type 1 and 1, which is the only particle type considered in your simulation. If you have more particle types, then you will have to define coefficient for all the combinations of the particle interaction types, otherwise LAMMPS will show an error.

In your code you have defined the conservative force constant a_{ij} or which is also represented as A as 25.0.

To ensure the proper thermo-stating you will need to define a good σ_{dpd} (please do not confuse with the representation of length often by the same symbol) coefficient. In LAMMPS you define γ, which is in your example script defined as 4.5. And the σ is related to γ by expression $\sigma = \sqrt{2k_BT\gamma}$. So for a defined $\gamma = 4.5$, you have σ implemented as $\sigma = 3$. For larger force coefficient A (or a_{ij})there will happen more caging and system will take more time to equilibrate. Therefore you will need to chose higher value of σ. For example, for the force constant $A = 25$ an appropriate value of σ is 3. In other words in LAMMPS you define the corresponding $\gamma = 4.5$ as shown in the example code.

Cut off length in your example code is defined as 1.0. It is an usual practice to make length scale calibration easy. It should always be less than **`rcD`**.

Steps to follow:

Please follow the following step to extract the ρ vs p (pressure) data. The mathematical relation between the two quantities are often refered as EOS (equation of state). Using the same data you can solve the related problems in your exercise.

1. Run the script file `e1_rho_and_pressure.in`. All students are advised to consider the a unique random force constant A (or a_{ij}) varying between the range $25 - 75$. Any copy paste will be punished by reduction of marks.

2. To set the different ρ, student can either change the box length (script line `variable L equal 10`). Or can change the number of particles created in the box by the command line `create_atoms 1 random 3000 12456 your_simBox`. As $\rho = N/L^3$ (where the value 3000 is the number of particles). Changing the box would be wise decision as increasing the number of particles will increase the simulation time.

3. For each value of ρ, students will simulate again and again and observe the terminal where the pressure is being pritent. It should convege to some value (a little fluctuation will be there but its fine). After sufficient simulation time take any value of the pressure, and it will be your pressure for that ρ value which you implemented in your system by altering simulation box or number of particles.

4. Once you get a set of ρ vs p value by setting different ρ and running the code for a particular force constant A. You can paste this value in the two column in the file `e1_rho_vs_pressure.txt`, where in each row there will be the set of rho and the corresponding pressure value.

5. After creating the text file students should open the python script `e1_plotter.py`. In the line, `a = 25` within the same python script, please set the force constant value to the value which you use in your own LAMMPS script. Then run the python code to generate the plot. From the plot you can calculate the value α, which is a constant and equal to $p - \rho/(A\rho^2)$.

3 Instructions for running the codes stored in folder: e1_binary_mixture

In this section, you will set the relation between the simulation parameters and the theoretical quantities which governs the evolution of the system of polymers (in the binary mixture). Here are some customization and reconfiguration of your LAMMPS code `e1_flory_parameter.in`.

System configuration:

To make a binary system of polymers, you will add another type of beads in the system and also define the iterations among the different types of beads, which is explained below in detail.

1. **Creating memory space for the two types of particles:**

```
### Define simulation box
variable     Lx equal 20
variable     Ly equal 10
variable     Lz equal 10

region       your_box_total block 0 ${Lx} 0 ${Ly} 0 ${Lz}
create_box   2 your_box_total  # Binary system with 2 particles types
```

In this relevant section of your LAMMPS script `e1_flory_parameter.in`, you are creating a rectangular box, which is extended extra in the x-direction. The command `create_box 2 your_box_total` makes space (memory allocation) for the two types of the particles or beads.

2. **Creating two type of particles in the designated regions:**

```
# Regions for particle distribution
region       region_A block 0 10 0 ${Ly} 0 ${Lz}
region       region_B block 10 20 0 ${Ly} 0 ${Lz}

# Create particles in each region
create_atoms 1 random 3000 12345 region_A  # Type A: atoms in region A
create_atoms 2 random 3000 67890 region_B  # Type B: atoms in region B
```

Now you define two section in the box along x axis. One on left from the length $0-10$ and another from the $10-20$. You create atom type 1 in the region A and type 2 in the region B. But remember, particles are not restricted to live in any region, they are free to move throughout the whole simulation box defined by the `create_box` command. We are doing this, as in the case if system will phase separate it will separate randomly in the simulation box, and it would be difficult to first track those regions and then count the number of a particular species present in the region just to calculate the density of the species in that region. And if we have already separated, then in a phase seperated configuration particles will not mix and we can count the density of the particle in that region easily, as the separated region often does not shift during the simulation.

Note: For example, if you are defining a region with extension beyong the region used in the `create_box` command. Then while generating the particles within the specified regions, those particle will be excluded which will come out of the box extension. So you will get less number of particles generated, however LAMMPS will not show any error and the code will start running with the same less number of generated particles. So be aware while redefining the regions. It must be a subset of the `create_box` command region.

3. **Defining the pair coefficient among the different types of particles:**

```
### Define DPD pair style
pair_style   dpd ${T} ${rcD} 3854262
pair_coeff   1 1 25.0 4.5 1.0  # A-A interactions
pair_coeff   1 2 45.0 4.5 1.0  # A-B interactions (adjust as needed)
pair_coeff   2 2 25.0 4.5 1.0  # B-B interactions
```

Now, it comes to the most relevant part, which you will alter during your exercise. So in the script, you can see that the pair coefficient for all the combination, for the two kinds of particles are defined by using their particle type id. (particle type id is not same as the particle id defined in the LAMMPS, a particle type id represents all the particles of the same kinds present in the system). The coefficient A, which represents the strength of the repulsive force will decide the mixing and the de-mixing of polymers in your system. If we set the value $A_{11} = A_{22}$, then by using the mean field expression, we find out that the Flory parameter χ is linearly related with the quantity $A_{12} - A_{11}$. So for more de-mixing, you will alter A_{12} towards the greater value (which will eventually be equivalent to increasing the χ value), specified in your command `pair_coeff 1 2 45.0 4.5 1.0`, (which is 45 in the given example script).

Steps to follow:

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

1. Run the LAMMPS script `e1_flory_parameter.in` after setting desired value of A_{ij} pair coefficient.
2. You can visualize the system using OVITO; just simply load the data file in OVITO and set the particle size and color etc. You will be taught this in your class and please remember or note down the steps during the class.
3. The script export all the data in an output file named `simulation_data.lammpstrj`.
4. Within the same folder (`e1_density_distribution.py`) you have been provided a python script, which will give you the value of density and x; the fraction of the density of type B particle in some phase separated region.
5. Use the value of the fraction of the density (x) and calculate the χ using the expression

$$\chi = \frac{\ln\left[(1-x)/x\right]}{1-2x}.$$

6. For different different value of $A_{12} - A_{11}$, where $A_{11} = A_{22} = 25$ is always kept constant, you can calculate the value of χ. So you have a list of *chi* vs $\Delta a = A_{12} - A_{11}$, data, which you can plot to get the value of the β in the given expression $\chi = \beta \Delta a$

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.