The `lammps_multistate_rods` library

User Manual

Eugen Rožić

May 19, 2022

1 Introduction

This library enables running hybrid MD-MC simulations in LAMMPS of interacting rods that can assume multiple states. The dynamics of the system, as a fast process, is the MD part of the simulation, thus enabling the full potential of LAMMPS to be used, while the changes in the (internal) states of the rods, as a slow process, can be implemented as batches of N MC moves every M MD steps (this makes the most sense in general, but the user can vary this arbitrarily). Each MC move is consisted of choosing a rod and a new allowed state for that rod at random and then accepting the new state of the system with probability

$$P_{i\to j} = \max\left\{e^{-(\Delta U+\Delta\mu_{ij})},\ 1\right\}, \tag{1}$$

where ΔU is the change in the interaction potential energy of the system and $\Delta\mu_{ij}$ is an energy penalty due to the change in the internal state of the rod.

The library provides a way to define a rod model which is then used to create the rods for the simulation. Each rod is consisted of a certain number of beads (particles) and **a state of a rod** is defined by the types of those beads, so a change in the state of a rod results in the change of the types of the beads (particles) a rod is consisted of. Since interactions in LAMMPS are defined through particle (bead) types this change of a rod's state results in a change in the interaction potential energy of the system and different dynamics.

1.1 Defining a model

A rod model is given by a textual configuration file and it defines the geometry and mass of the rods, all possible states a rod can assume, the interactions between rods and the transition penalties between different rod states.[1]

The configuration file has the syntax of a Python file that is restricted to one-line commands, with an exception of lines ending with a comma. The file is interpreted one line at a time in a protected environment in the process of creation of a `Rod_params` instance. Some variables have special meaning as parameters of the model and their values will be kept in the `Rod_model` instance, others can be defined for auxiliary purposes in the configuration file itself and will be discarded after a `Rod_model` instance has been created.
An example of a configuration file with all possible parameters explicitly set can be found in the `examples` directory in the library's GitHub repository (https://github.com/Saric-Group/lammps_multistate_rods).

Through the library each rod is implemented as a small, rigid molecule that has a body, consisted of body beads, and zero or more **patches**. A "patch" is a number of **interaction sites** (particles) in a line parallel to the rod axis that share some properties. There are no intra-rod interactions, i.e. interactions between particles/beads/interaction sites of the same rod are set to 0. All of the geometrical parameters at the user's disposal are depicted and described in figures 1 and 2.

Some of the features and restrictions of the model are:

- The number of body beads, patches and beads in each of the patches are also set in the configuration file, although implicitly through the `state_structures` parameter that also defines the types of all the beads in each of the states;

[1] All other interactions, e.g. between rod beads and other non-rod particles in the system, have to be given explicitly to LAMMPS in the program/input script.

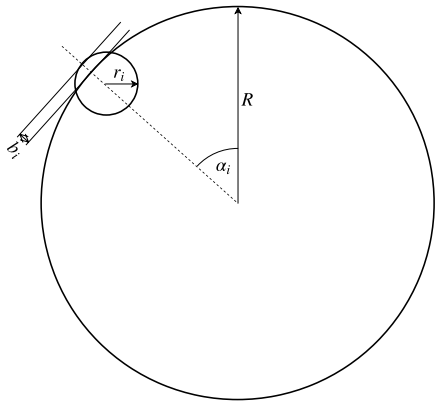

Figure 1: A front-to-back view of an example of a rod model with one patch. In a configuration file, R corresponds to the `rod_radius` *parameter and α_i, r_i and b_i correspond to the i^{th} elements of the* `patch_angles`, `patch_bead_radii` *and* `patch_bulge_out` *tuples.*

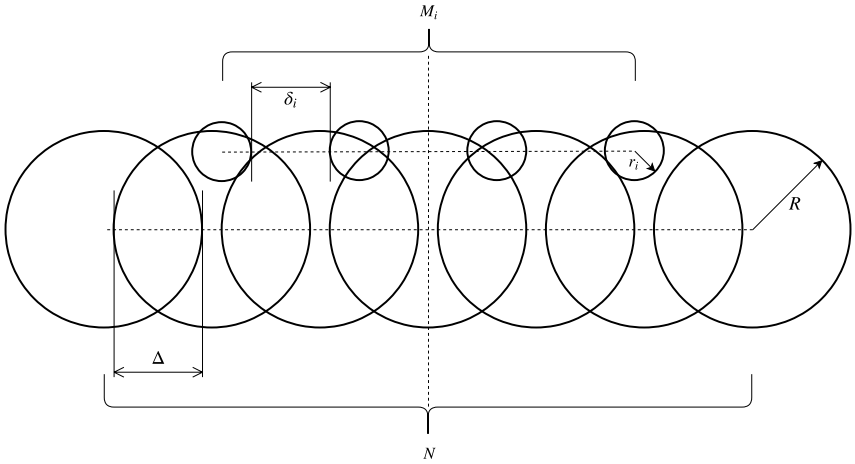

Figure 2: A side view of an example of a rod model with one patch. In the library, Δ (the overlap between body beads) is calculated from the `rod_length` *(from tip to tip) and* `rod_radius` *(R) parameters and the number of body beads N, which is implicitly given with the* `state_structures` *parameter; δ_i (the distance between interaction sites in a patch) corresponds to the i^{th} element of the* `patch_bead_sep` *tuple in the configuration file.*

- All states have to have the same geometry (numbers and positions of beads), but some of them can be made *inactive* (e.g. 0 interaction strength, or just volume exclusion) in some states;
- The parameters of an interaction are its **type** (i.e. potential function), **strength** and **range**, and are given separately for each pair of bead types. The **range** of an interaction is taken to be the distance between the *surfaces* of two beads at which the interaction vanishes, where the *surface* of a bead is taken to be the radius distance from its center/location.

1.2 Running a simulation

This library is written in Python 2.7 and uses the Python wrapper of the LAMMPS library interface (`PyLammps`) to communicate with the LAMMPS program. In order to use it the user first has to make a Python script that imports the `lammps` module and create an instance of `PyLammps` which they will afterwards use in essentially the same way as in a standard textual input script.
Secondly, the user has to construct a `Rod_model` object using a `Rod_params` instance which was previously instantiated and used to parse the model configuration file. This `Rod_model` object holds all data from the configuration file and a lot of additional, inferred ones that instances of the `Simulation` class use, but can also come in handy to the user. These two objects (a `PyLammps` and a `Rod_model` instance) can

then be used to instantiate a `Simulation` object.

The idea of the library is for it to be used to create and manipulate the (states of the) rods as independently as possible of any other particles or features in LAMMPS. To this end the `Simulation` class offers various convenience methods for setting up and querying the LAMMPS simulation, creating the rods and (un)setting fixes that define the behavior of the rods.
All other elements and features of the simulation (not related to the rod objects) can be created and set independently of the library using the `PyLammps` object, as one would normally do in a LAMMPS (Python) input script.

After a `Simulation` object is created and some basic properties of the simulation (e.g. units, dimensions, boundaries etc.) set, the `setup` method should be called with some information about all the other non-rod LAMMPS elements (particles, interactions, etc.) that are or will be used in the simulation. This way the library takes care of everything else for the user and it is ensured no conflicts between the rod and non-rod aspects of the simulation arise.
Next the rods can be created in various (documented) ways with the `create_rods` method, but also the *gcmc* fix of the `set_state_concentration` can be used for such a purpose too.
The library offers methods to set up 3 main types of behaviors of rods through LAMMPS fixes, and those are:

1. `set_rod_dynamics` - sets the dynamics of ALL the rods through a *rigid/small* fix of a given ensemble (if the user wants different dynamics for different groups of rods they will have to do it "by hand", but the contents of this method can be a good guide);

2. `set_state_transitions` - sets up the change-of-state MC moves through a *change/state* fix[2] using the information from the rod model (allowed transitions and corresponding penalties)

3. `set_state_concentration` - sets up a *gcmc* fix (only for insert/delete moves) to keep (approximately) constant concentration of rods of a given state in the simulation (currently available only for one state because of the limitations of the *rigid* fixes)

The user can also define additional behaviors relating to the rod objects outside of using the methods of the library / `Simulation` object, but should be very careful with that and familiarise themselves beforehand with the inner workings of the (relatively well documented) library classes.

A demonstration of all of this and an example of how the library can be used can be found in the `examples` directory in the form of a console application (`simulate_nvt.py`).

Technical limitations and development points

1. Energy minimization algorithms from LAMMPS cannot be used on the rods because they are defined as rigid objects. However, random initial configurations can be generated in multiple other ways:

    - using the *random* option of the `create_rods` method (advisably with the *overlap* and *maxtry* keywords)
    - using the *gcmc* fix of the `set_state_concentration` method in an initial "equilibration" run with state-changing disabled, etc.

[2]This is a fix developed by me and can be found in my personal fork of LAMMPS, it is not yet available in the official LAMMPS, but it probably will be in the near future