The `lammps_multistate_rods` library

User Manual

Eugen Rožić

June 19, 2019

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_model` 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 rigid body 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 along 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.

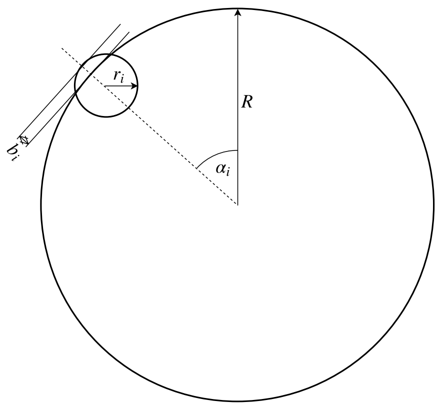

Figure 1: A front-to-back view of an example of a rod model with one patch. In a configuration file, R corresponds to the `rod_radius` *parameter and α_i, r_i and b_i correspond to the i^{th} elements of the* `patch_angles`*,* `patch_bead_radii` *and* `patch_bulge_out` *tuples.*

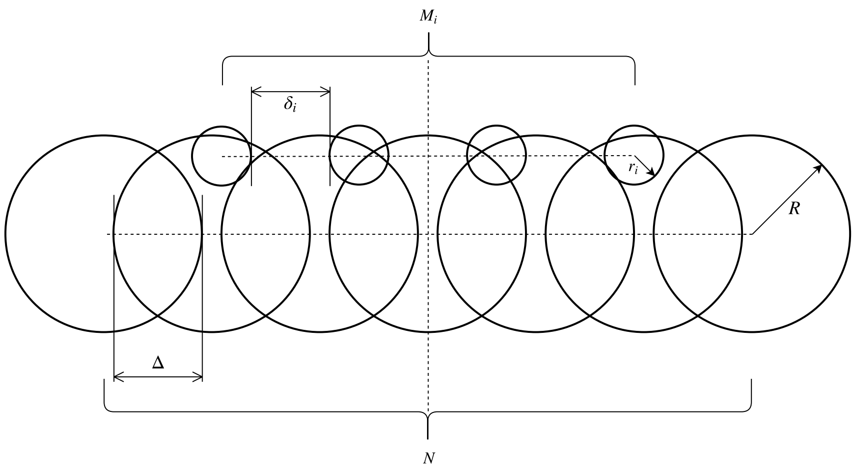

Figure 2: A side view of an example of a rod model with one patch. In the library, Δ (the overlap between body beads) is calculated from the `rod_length` *(from tip to tip) and* `rod_radius` *(R) parameters and the number of body beads N, which is implicitly given with the* `state_structures` *parameter; 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 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 `Lammps` (or `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 the model configuration file they previously defined. This 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 `Lammps` and a `Rod_model` instance) can then be used to make 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 a `Simulation` object needs to be instantiated and `setup` with some information about all the other LAMMPS elements (particles, interactions, etc.) that are or will be used in the simulation. After that the object's methods (e.g. `create_rods`, `set_rod_dynamics`, `state_change_MC`) can be used to create rods, set their dynamics and manipulate their states. In the meanwhile LAMMPS can be used normally (through Python) to manipulate in any which way any other particles or properties of the system.

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. It is impossible to delete rods after they are created ;
2. Consequently to 1., a grand-canonical ensemble cannot be established (`fix gcmc` can't be used) for any of the rod states. However, a near-constant density of rods in a certain state might be achieved by simply adding new rods in that state (if that state gets depleted during the simulation);
3. Currently MPI functionality is not available, a simulation with rods can only be run on a single core (this is probably fixable without major changes to the library);
4. Energy minimization algorithms from LAMMPS cannot be used on the rods because they are defined as rigid objects. This makes random initial configurations hard to generate.