molsim tutorial

J.S. Hansen

Februaray 2022, v. 0.9

1 Introduction

molsim is a GNU Octave/Matlab package for molecular dynamics simulation library. molsim supports simulations of

- Standard Lennard-Jones systems (solid, liquids, gasses, etc)
- Molecular systems with bond, angle, and torsion potentials
- Confined flow systems, eg., Couette and Poiseuille flows
- Charged systems using shifted force and Wolf methods
- Dissipative particle dynamics systems
- and more

The package also supports a series of run-time sampling functionalities.

molsim is basically a wrapper for the seplib library, which is a light-weight flexible molecular dynamics simulation library written in ISO-C99. The library is CPU-based and offers shared memory parallisation; this parallisaton is supported by the molsim package. The algorithms used in seplib is based on the books by Allen & Tildesley, Rapaport, Frenkel & Smith, and R. Sadus, see Ref. [**?**].

In this text

```
>>
```

indicates GNU Octave or Matlab command prompt. This

```
$
```

indicates the shell prompt.

2 Installation

2.1 GNU Octave

GNU Octave's package manager offers a very easy installation. From

`https://github.com/jesperschmidthansen/molsim/`

download and save the current release `molsim-<version>.tar.gz` in a directory of your choice. Start GNU Octave and if needed change directory to the directory where the file is saved.

```
>> pkg install molsim-<version>.tar.gz
```

Check contact by

```
>> molsim('hello')
Hello
```

In case this fails, check the path where molsim is install by

```
>> pkg list molsim
```

If the path is not in your GNU Octave search path add this using the `addpath` command.

2.2 Matlab

From

`https://github.com/jesperschmidthansen/seplib/`

download and save the current release `seplib-<version>.tar.gz` in a directory of your choice. Unpack, configure and build the library

```
$ tar zxvf seplib-<version>.tar.gz
$ cd seplib
$ ./configure
$ make
$ cd octave
```

To build the mex-file enter Matlab

```
$ matlab -nodesktop
```

Then build the

```
>> buildmex
```

Depending on the system this will build a molsim.mex¡archtype¿ file. You can copy this file to a directory in your Matlab searce path.

3 First quick example: The Lennard-Jones liquid

Listing 1 shows the simplest script simulating a standard Lennard-Jones (LJ) system in the micro-canonical ensemble where number of particles, volume, and total energy is conserved.

Listing 1

```
% Specify the LJ paramters
cutoff = 2.5; epsilon = 1.0; sigma = 1.0; aw=1.0;

% Set init. position and velocities 10x10x10 particles in box
% with lengths 12x12x12. Configuration stored in start.xyz
molsim('set', 'lattice', [10 10 10], [12 12 12]);

% Load the configuration file
molsim('load', 'xyz', 'start.xyz');

% Main loop
for n=1:10000

  % Reset everything
  molsim('reset');

  % Calculate force between particles of type A (default type)
  molsim('calcforce', 'lj', 'AA', cutoff, sigma, epsilon, aw);

  % Integrate forward in time
  molsim('integrate', 'leapfrog');

end

% Free memory allocated
molsim('clear');
```

Listing 1 is not very useful as no information is printed or saved. Inside the main loop you can add the command

```
if rem(n,100)==0
  molsim('print');
end
```

to print current iteration number, potential energy per particle, kinetic energy per particle, total energy per particle, kinetic temperature, and total momentum to screen every 100 time step.

More information can be retrieved from the simulation, e.g., get the system energies and pressure

```
[ekin, epot] = molsim('get', 'energies');
press = molsim('get', 'pressure');
```

and particle positions and velocities

```
x = molsim('get', 'positions');
v = molsim('get', 'velocities');
```

In general, the `molsim` interface is on the form

```
molsim(ACTION, SPECIFIER, ARGUMENTS)
```

where the action can be `get`, `integrate`, and so on, the specifier is a specification for the action, and arguments are the arguments for the specifier.

3.1 NVT and NPT simulations

Often you will not perform simulations in the microcanonical ensemble, but under a desired temperature or/pressure. One way to achieve this with `molsim` is to use simple relaxation algorithms. To simulate at temperature, say 2.2, you call the action 'thermostate' after the integration step

```
molsim('thermostate', 'relax', 2.2, 0.01);
```

The last argument is the relaxation parameter; the higher value the faster relaxatio. Notice that too large values makes the system unrealistically stiff. The best value is optimed via trail-and-error.

To simulate at pressure, say 0.9, you call the action 'barostate' after the integration step,

```
molsim('barostate', 'relax', 1.5, 0.01);
```

The choice of relaxation parameter is again a matter of the system. You can use the two relaxation actions in the same simulation mimicking an NPT system. The barostate works by changing the system box length in the z-direction only; this is practical when doing hydrodynamic sampling, see later.

4 The force field

$$
\begin{aligned}
U &= U_{\text{LJ}} + U_{\text{Coloumb}} + U_{\text{bonds}} + U_{\text{angles}} + U_{\text{diheadrals}} + U_{\text{lattice}} &(1)\\
&= \frac{1}{2}\sum_{\text{bonds}} k_n(r_{ij} - l_b) + \frac{1}{2}\sum_{\text{angles}} k_m(\cos(\theta) - \cos(\theta_m))^2 &(2)
\end{aligned}
$$

5 Molecular systems

6 Sampling

7 The two parallisation paradigms