molsim tutorial

J.S. Hansen

Februaray 2022, v. 0.9

1 Introduction

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

- standard Lennard-Jones systems,
- 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

```
>>
```

symbolises the GNU Octave/Matlab command prompt. This

```
$
```

symbolises 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.
Then type

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

to install the package. 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 search 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. Velocities set to default.
% Configuration stored in start.xyz.
molsim('set', 'lattice', [10 10 10], [12 12 12]);

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

% Main mol. simulation loop - 10 thousand time steps
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 - use leapfrog alogrithm
  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.

Information can also be stored into variables for further analysis. For example, to 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`, `calcforce`, and so on, the specifier is a specification for the action, eg., `lj` specifýing that the force is a pair-wise Lennard-Jones force, 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 and/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'` with specifier `'relax'` after the integration step

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

The last argument is the relaxation parameter; the higher value the faster relaxation. 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', 0.9, 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 (an-isotropic scaling); this is practical when doing sampling as two directions are fixed.

4 The molsim force field

molsim supports simulations of confined, charged, and molecular systems. In general, the interaction molsim force field is defined from the potential functional form

$$U(\mathbf{r}_i, r_{ij}, \ldots) = U_{\text{lattice}} + U_{\text{vWaals}} + U_{\text{coloumb}} + U_{\text{bonds}} + U_{\text{angles}} + U_{\text{dihedrals}} \quad (1)$$

The first term allows for a simulation of a fictitious fixed crystal arrangement, where the particles/atoms are tethered around a pre-set lattice site. The potential function is a harmonic spring type

$$U_{\text{lattice}} = \sum_{\text{sites}} \frac{1}{2} k_0 (\mathbf{r}_i - \mathbf{r}_0)^2 \quad (2)$$

where k_0 is the spring constant, $\mathbf{r}_i$ is the position of particle/atom i, and $\mathbf{r}_0$ is the lattice site. The default lattice site positions are the initial particle positions.

The short ranged van der Waals pair interaction is given via the standard Lennard-Jones potential

$$U_{\text{vWaals}} = \sum_{i,j \text{ pairs}} 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - a_w \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] . \quad (3)$$

Here r_{ij} is the particle distance, ϵ and σ define the characteristic energy and length scales. The parameter a_w determines the weight of the attractive second term in the potential function.

The Coulomb potential is

$$U_{\text{coloumb}} = \sum_{i,j \text{ pairs}} \frac{q_i q_j}{r_{ij}} . \quad (4)$$

This long ranged interaction is evaluated using approximative shifted-force or Wolf methods; this can be specified. These two implementations do not apply to confined systems.

Particle bonds are model via the harmonic spring potential

$$U_{\text{bonds}} = \sum_{\text{bonds}} \frac{1}{2} k_s (r_{ij} - l_0)^2 \tag{5}$$

k_s is the spring constant and l_0 is the zero force bond length. Note, currently molsim does not support rigid bonds.

The angle potential is the cosine squared potential

$$U_{\text{angles}} = \frac{1}{2} \sum_{\text{angles}} k_\theta (\cos(\theta) - \cos(\theta_0))^2 \,, \tag{6}$$

where k_θ is the force amplitude, and the zero-force angle θ_0. See Fig. **??** for the angle definition.

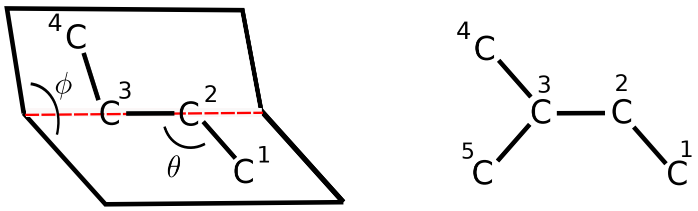

Figure 1: sds

Finally, the dihedral angle potential is the Ryckaert-Belleman potential

$$U_{\text{dihedral}} = \sum_{\text{dihedrals}} \sum_{n=0}^{5} c_n \cos^n(\pi - \phi) \,. \tag{7}$$

Here c_n are the six Ryckaert-Belleman coefficients, and ϕ is the dihedral angle, see Fig. 2. Two illustrative examples are when $c_n = 0$ except for $n = 1$:

1. If $c_1 > 0$ then the minimum energy dihedral angle is $\phi = 0$; this is illustrated in the right-hand figure in Fig. 2 with dihedral defined by the 1-2-3-5 bonds. Useful for closed molecular ring structures.

2. If $c_1 < 0$ then the minimum dihedral angle is $\phi = \pi$; this is illustrated in the right-hand figure in Fig. 2 with dihedral defined by the 1-2-3-4 bonds. Useful for branched molecular structures.

Below a more complex example is given.

Importantly, you can have different types of particles with different charges, different bonds, angles, and dihedrals, all determined from the interaction parameters. It is thus possible to simulate mixtures, highly complex molecules, etc.

5 Molecular systems: Toluene

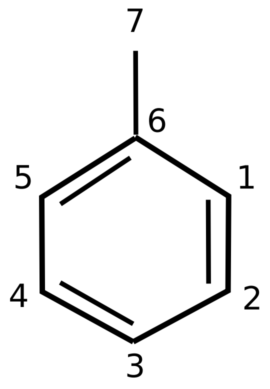

Figure 2: sds

```
% Simulation parameters
temp0 = 4.0; dens0 = 1.95;

dt  = 0.001; nloops = 200000;

% Intra-molecular parameters
bondlength_0 = 0.4; bondlength_1 = 0.38; springconstant = 48910;

bondangle = 2.09; angleconstant = 1173;

torsionparam_0 = [0.0, 133.0, 0.0 0.0 0.0];
torsionparam_1 = [0.0, -133.0, 0.0 0.0 0.0];

% Load positions, set temp, remove intra-molecular pair-interaction etc
molsim('load', 'xyz', 'start_toluene.xyz');
molsim('load', 'top', 'start_toluene.top');
molsim('set','timestep', dt);
molsim('set', 'temperature', temp0);
molsim('set', 'exclusion', 'molecule');

% Main loop
for n=1:nloops
  molsim('reset')

  molsim('calcforce', 'lj', 'CC', 2.5, 1.0, 1.0, 1.0);

  molsim('calcforce', 'bond', 0, bondlength_0, springconstant);
  molsim('calcforce', 'bond', 1, bondlength_1, springconstant);

  molsim('calcforce', 'angle', 0, bondangle, angleconstant);

  molsim('calcforce', 'torsion', 0, torsionparam_0);
  molsim('calcforce', 'torsion', 1, torsionparam_1);

  molsim('thermostate', 'nosehoover', 'C', temp0, 10.0);
  molsim('integrate', 'leapfrog');
end
```

6 Dissipative particle dynamics (DPD)

7 Sampling

8 The two parallisation paradigms