CASHEW tutorial

Teemu Hynninen
Tampere University of Technology, Finland
Helsinki University of Technology, Finland
March 22, 2011

Introduction

This is a short tutorial for the molecular dynamics simulation tool CASHEW (Coarse Approach Simulator for Hydrogen-bonding Effects in Water). To go through the tutorial, you should have this tutorial, the source code for CASHEW or a working executable, the CASHEW manual, and the example files provided with the tutorial.

Compile CASHEW according to the instructions in the manual. Though not necessary for this tutorial, it is always good to test compiler optimization possibilities and use the most efficient version possible, at least for production runs. Examples 2 and 3 provide systems that may be used for testing compiler efficiency. In the following, we assume that the name of our executable is `cashew`.

The examples listed here are increasingly more complex, so it is recommended to go through them in order. However, none of them actually require the previous ones to be completed. The input files provided with the tutorial are split to folders according to the examples: Files needed for example 1 are in folder `t1`, for example 2 in `t2`, and so on. To do the exercises, copy the files to the directory where you have your CASHEW executable and run the code there.

1 MB basics

topics:	MB force-field, running CASHEW
new input:	`<control>`, `<particles>`, `<velocities>`
files:	`tutor1.mb`: `tutor1.out`, `tutor1.xyz`

We'll start the tutorial by taking a look at the basic structure of the input and the properties of the MB model. Find the file `tutor1.mb` and open it in a text editor. You should find four blocks defined: `<control>`, `<particles>`, `<velocities>`, and `<mb-model>`. The last one, `<mb-model>`, defines values for the physical parameters used by the MB-model, and you should not touch it at this point. The block `<control>` defines the length of the simulation, the time step used for numeric integration, an output frequency for xyz-data and the amount of general output. The block `<particles>` has two lines with labels "MB", meaning that the simulation will include only two MB molecules. Finally, `<velocities>` has two lines defining the initial velocities and angular velocities for these molecules.

Run the simulation with the command `cashew tutor1`. You should get three new files from the simulation: `tutor1.out`, `tutor1_end.mb`, and `tutor1.xyz`. The `.out` file is the main output file. In the beginning of the file, you'll find a more detailed version of the input you just gave to the program: the default values of the various parameters that were omitted have been written explicitly here. There is other data available as well. Change the numbers following the tag `verbose` in `<control>` (or remove it completely) and rerun the simulation to see how you can control the amount of output.

The file `tutor1.xyz` contains the system geometry, recorded every 5 fs. Open it with some visualization tool to see an animation of the movement of the molecules. You should see the distance between the two molecules oscillating: initially the molecules were 3.5 Åapart while their equilibrium distance is about 2.8 Å. They also wiggle about as the two closest dangling bonds attract each other and the others try to find dihedral angles of 60°.

Try changing the initial conditions of, say, molecule no.2 to see how the behavior changes. Examine especially how the orientation of the molecules is defined in `<particles>` and, with the help of the manual, try to figure out how to e.g. turn the molecules by 180° around a coordinate axis. Note that if you remove or comment the part defining the orientation, the program will assign a random orientation for the molecule. Finally, you can uncomment the

tag `thermo` to include friction in the simulation. This will slow down the oscillations until the molecules stop in an equilibrium configuration.

2 Ice cube

topics:	lattice structure (ice), thermalizing, statistics recording, Langevin thermostat
new input:	`<cell>`, `<statistics>`
files:	`tutor2.mb`: `tutor2_stats.out`, `tutor2.xyz`

In this example, we simulate ice and take a look at data gathering. Find and open the file `tutor2.mb`. There should be the tag `run test`, meaning that we want a test run, where we only check the system initialization. Run the file and look at the produced geometry by visualizing the resulting xyz file.

You should find a small supercell for hexagonal ice-I. Can you recognize the structure? Also, can you see which line in the `<particles>` block creates which molecule? This cell is by far too small for simulations though, so we need to expand the system and introduce periodic boundaries. Without a `<cell>` block, the system has free bounds. Uncomment the block in the input to apply the periodic boundaries. Then rerun the initialization and look at the system again.

The periodic boundaries are not visible in the xyz file, but you'll also see that the supercell is larger than before, containing 360 molecules instead of 8. This is because the `<cell>` block also told CASHEW to multiply the original 8 particle cell $3 \times 5 \times 3$ times (in x, y and z directions, respectively). Try changing these multipliers to see how the supercell size is controlled. Also take a look at the file `tutor2_start.mb`, which was created by the tag `writeinp startend`. There, you'll find the coordinates of all the molecules written out explicitly. This is a valid input file, so if we wished to make small changes to the ideal system, we could easily edit this file and run it instead. We won't do it now, though.

Return to the ideal 360 particle system and change `run test` to `run full` (or remove it) and rerun the system for a full simulation. This may take a few minutes depending on your computational setup. You can try running a larger system as well, but the simulation time will naturally be longer, so try the smaller system first. You'll see that the simulation resulted in an output file named `tutor2_stats.out`. Which contains data in three columns. This output was requested by the `<statistics>` block in the input, which contained the lines `interval 5.0`, `time`, `temperature` and `energy`. The first line commands CASHEW cashew to append the stats file after every 5 fs of simulation. The other three demand that simulation time, system temperature and system total energy are to be written in the file — hence the three columns.

Plot the system temperature and energy as a function of simulation time. Although the initial temperature was 400 K, you should see the temperature quickly dropping to somewhat above 200 K. This is because the initial geometry is close to the potential energy minimum, and roughly half of the initial kinetic energy is transformed to potential energy. The total energy is constant though, with only small fluctuations due to numeric integration error. If you wish, you may change the time step in numeric integration and see how the amplitude of energy fluctuations changes. If you uncomment the command `start 200.0`, the statistics recording will not start at the beginning of the run but after 200 fs instead and the initial rapidly changing part of the simulation will be cut from the stats file.

Next, uncomment the tag `thermo langevin 0.03` to change from a microcanonical simulation to Langevin dynamics with a friction coefficient $\gamma = 0.03$ fs^{-1} and recomment or remove the `start 200.0` command again. Rerun the simulation (you may want to save the previous results with another name or rename the input file so that the new output files have different names; otherwise they will be overwritten). Plotting the temperature and energy now should show very different behavior. The thermostat will push the temperature towards the required value of 400 K and keep it there once it has been reached. This naturally means that energy is not conserved as in the microcanonical case. Try changing the value of the friction coefficient and check how it changes the rate of temperature climb. (Note though, that you should always have $\Delta t \ll 1/\gamma$.)

3 Collision

topics:	non-equilibrium events, atomic particles, constraints
new input:	`<elements>`, `<potentials>`, `<constraints>`
files:	`tutor3_a.mb`, `tutor3_b.mb`: `tutor3_*.xyz`

In this exercise, we simulate a dynamic event, a particle impact, and see how particles other than MB molecules can be included. There are two prepared input files for this example: `tutor3_a.mb` and `tutor3_b.mb`. Start with `tutor3_a.mb`.

Examine the block `<cell>`: you should be able to guess that the system is a slab since it is periodic in two dimensions. The `<particles>` block is rather big in this case, but note how the first particle is not an MB molecule but something called `ptile` (projectile). What it is, is defined in blocks `<elements>` and `<potentials>`:The former one, `<elements>`, is just a list of different atomic particles (other than MBs) present in the simulation, along with their masses. In this case, there is only one, namely `ptile`. The latter block, `<potentials>`, defines how the atomic particles interact with MBs and each other. In this case, `ptile` only interacts with MBs — but there is only one such particle in the system so any `ptile`–`ptile` interaction would be meaningless. The interaction is described by a shifted Lennard-Jones (hard-sphere) potential, as defined by the tag `hard`.

Run the simulation and animate the resulting xyz file. You should see the projectile hitting the thin slab of ice, kicking some MB molecules about. Since the projectile was a hard-sphere, most of its momentum is initially given to the MB molecules closes to the point of impact. You can rerun the simulation with different initial velocities of the particle by editing the block `velocities`. If you give the projectile enough momentum, it will break through the slab creating a hole on its way.

The block `<potentials>` also has a commented tag `pow`, which defines a power-law potential. Compared to the hard-sphere potential above it, this potential decays slowly. Remove or comment out the hard-sphere interaction, uncomment the power-law, and rerun the simulation. You should see that for this interaction, the impact is absorbed by a larger group of MB molecules. (If you run a longer simulation, you should be able to see the ice sheet being left oscillating after the impact.)

Now, let us move to the other input file, `tutor3_b.mb`. It is similar to the previous input except for the block `<constraints>`. In `tutor3_a.mb`, only a few molecules had constraints acting on them: the tag `well` defines a harmonic well potential that pulls the constrained molecules towards their original positions. This was done so that the impact would not send the entire slab moving. Now, however, you should find that a large group of molecules if pinned to their initial positions using the `frozen pos` tag. This constraint fixes the position of a particle but allows a molecule to rotate. If you examine the list of particles, you should see that the ones constrained are at the bottom layer of the slab. As these molecules are fixed to their ideal bulk ice positions, this setup can be thought to mimic a semi-infinite block of ice.

Rerun the simulation and compare to the previous results. You should find that in this case, the ice is not as easily damaged as in the previous simulations where a thin film was bombarded. If you increase the energy of the projectile too much, however, you'll see that some molecules may still penetrate the slab. This basically means that you should have a bigger system to properly simulate these impacts. (Here, the system is kept small in order to have reasonable run times for tutorial purposes.)