LJ-Canonical Tutorial

PUE Advanced Computational Physics
University of Vienna - Faculty of Physics

Metropolis Monte Carlo simulations of the Lennard-Jones system

The Lennard-Jones (LJ) potential,

$$u(r) = 4\varepsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right], \tag{1}$$

has proven to be a good model for the effective pair interaction between pairs of noble gas atoms or molecules with a spherical symmetry. The program package lj-canonical, originally written by Prof. Martin Neumann, implements a Metropolis Monte Carlo (MC) simulation of this system under conditions of constant particle number, volume, and temperature (NVT ensemble). The package is available in both C as well as Fortran. Please consult the README file for information on how to compile the code.

The simulation code consists of a number of smaller helper programs and the main Monte Carlo simulation. Usage is the same for both versions, but binary files are not compatible. The following binaries are available.

- **gmclj** creates a binary input file with a random (gas) configuration.
- **lmclj** creates a binary input file with an fcc lattice configuration.
- **mclj** is the main simulation code, which will read an input file, and write an (also binary) output file of the same format.
- **zmclj** is used to reset all accumulated averages after an initial equilibration phase.
- **amclj** reads an output file and prints results in ASCII form, such as accumulated averages or the pair correlation function g(r).

In the following, we will set up and run a simulation and calculate the pair correlation function. To make everything run fast, we will use a very small system of only $N = 256$ particles. We will also use a reduced temperature of $T^* = 1.2$ and a reduced density of $\rho^* = 0.8$.

We start by creating a gas configuration using *gmclj*, see the example command line output below. It is advisable to do everything in a separate directory from the main code. In the following, we assume that we are in a directory *tutorial*, which is inside the main source code directory where all the utilities are. Enter the desired values, and hit return to confirm. *disp* is the maximal particle displacement in a Monte Carlo move. Use 0.3 to get an overall acceptance rate of 30% to 50%. *dr* is the grid spacing for the calculation of $g(r)$. You can use whatever you see fit here, but 0.01 is a good choice. The final three parameters control how long the simulation will run. In a Monte Carlo simulation of N particles, one *pass* is typically defined as N attempted single-particle moves. The parameter *ntskip* defines how many passes are performed between the collection of averages. *ntprint* only controls the print frequency to the screen. For now, use 1 for both. Finally, *ntjob* sets the total run time, start with a value of 100. Hit return to accept the default value for the input file name.

```
[user@PC tutorial]$ ../gmclj
              n=256
            rho=0.8
              t=1.2
           disp=0.3
             dr=0.01
         ntskip=1
 ntprint/ntskip=1
   ntjob/ntskip=100
          fname=[mclj_in.dat]

[user@PC tutorial]$ ../mclj
 1   4.60938e-01   2.51474e+08   8.04720e+08
 2   4.80469e-01   1.83327e+06   5.86739e+06
 3   4.41406e-01   6.69773e+05   2.14371e+06
 ...
```

After creating the input file, run the simulation by executing *mclj*. The program writes four columns of output: (MC) time, current acceptance ratio, current potential energy per particle, and current pressure. You will see that due to a purely random particle configuration, the initial energy and pressure are very high. During the simulation, these quantities will slowly approach equilibrated values. The program also periodically writes the current system configuration to an XYZ trajectory file, *traj.xyz*. This happens every time values are printed to the screen, so every $N \times ntskip \times ntprint$ single-particle moves. In a previous version of the program, the XYZ output frequency was higher (every $N \times ntskip$ single-particle moves). Feel free to tinker with the code if you think this is still too much output. You can even completely get rid of the XYZ output by commenting out / deleting the call of the *writeXYZ* function (*writexyz* subroutine in Fortran).

If you are on Linux or macOS, you can use

```
[user@PC tutorial]$ ../mclj | tee log.dat
```

to save the screen output in a log file *log.dat* for later plotting and analysis. For our final result, we do not want to use the averages accumulated in the initial run, so that is why we will use *zmclj.* in the next step. We can again simply hit return to accept default values, and only change *ntprint* (we do not want so much screen output) and *ntjob* (we want to run the simulation much longer):

```
[user@PC tutorial]$ ../zmclj
          infile=[mclj_out.dat]
                n=         256
             rho=[     0.80000]
               t=[     1.20000]
            disp=[     0.30000]
              dr=[     0.01000]
          ntskip=[           1]
  ntprint/ntskip=[           1] 100
    ntjob/ntskip=[         100] 5000
         outfile=[ mclj_in.dat]
```

Run the simulation again by executing *mclj.* This time, it will take a bit longer. Finally, we can use *amclj* to have a look at the results:

```
[user@PC tutorial]$ ../amclj
fname=[mclj_out.dat]

     n=       256
   rho=  0.80000
     t=  1.20000
  disp=  0.30000

    nt=         5000 (*     1)
  accr= 3.54852e-01
<U>/N=-5.36266e+00
  Cv/N= 2.43441e+00
     p= 1.96675e+00

Write g(r) to 'amclj.dat? [y] y
Write PDB format to amclj.pdb? [y] y
```

We see some parameters again, and, crucially, also results for average energy, heat capacity, and pressure. Note how the total simulation time is 5,000 (and not 5,100) passes due to the use of *zmclj.* If we want to run the simulation longer, we can also rename the output file *mclj_out.dat* to *mclj_in.dat*, and run *mclj* again. The simulation will then continue from the last configuration of the previous run.

The resulting pair correlation function $g(r)$ is shown in figure 1. As a next step, you can redo the tutorial with a lattice configuration. Make sure to choose temperature and pressure conditions within the solid phase, and then compare the resulting $g(r)$!

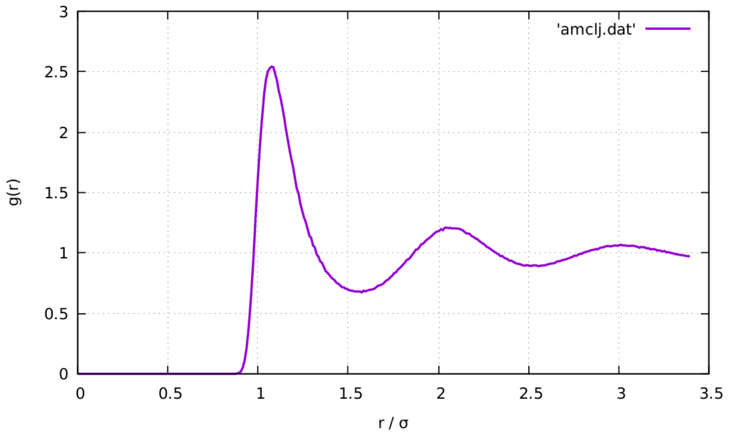

Figure 1: Pair correlation function calculated in this tutorial.