Worksheet 3: Molecular Dynamics 2 and Observables

April Cooper, Patrick Kreissl und Sebastian Weber

December 13, 2012
University of Stuttgart

Contents

1 Functions for temperature and pressure

We were supposed to implement `compute_temperature()` and `compute_pressure()`.

- `compute_temperature()` was implemented in the Python-part. Because it constits of very few operations it isn't time-consuming.
- `compute_pressure()` was implemented in the C-part. Here, you have to calculate a large sum in a large loop. Furthermore the access on the variable `verlet_list` is easier in the C-part than in the Python-part.

2 Running averages

In this section we had to calculate the running averages of the measured observables. The resulting plot can be seen below:

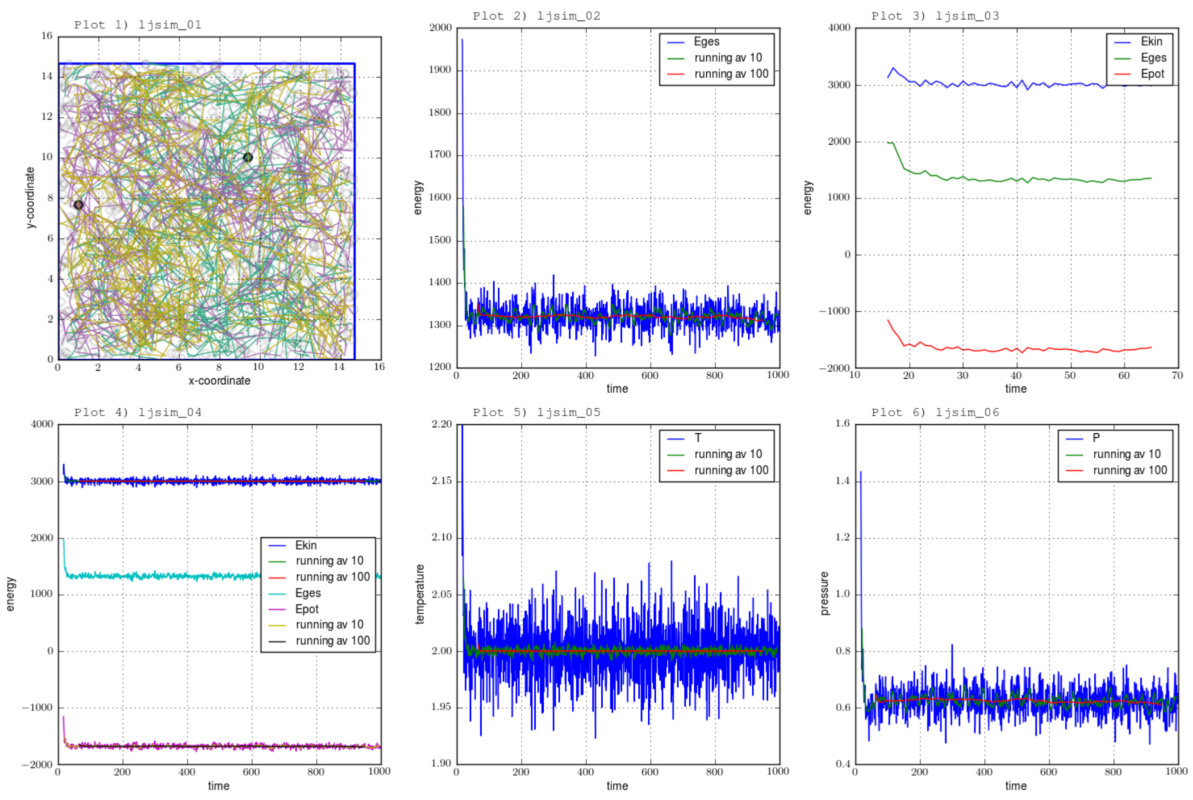


Figure 1: Observables up to t=1000 and their running averages

It can be observed that the observables fluctuate during the first time steps immensely and after some time they still fluctuate a bit, but their mean values become quite constant. This is due to the fact that first the system is equilibrating wherefore the observables fluctuate quite much, after this process the system is in equilibrium and therefore the observables stay constant. The time the system needs to be equilibrated is about $t_equi = 20$ time units. This can be easily seen as all the following graphs and especially all running averages showing a rather constant behavior .

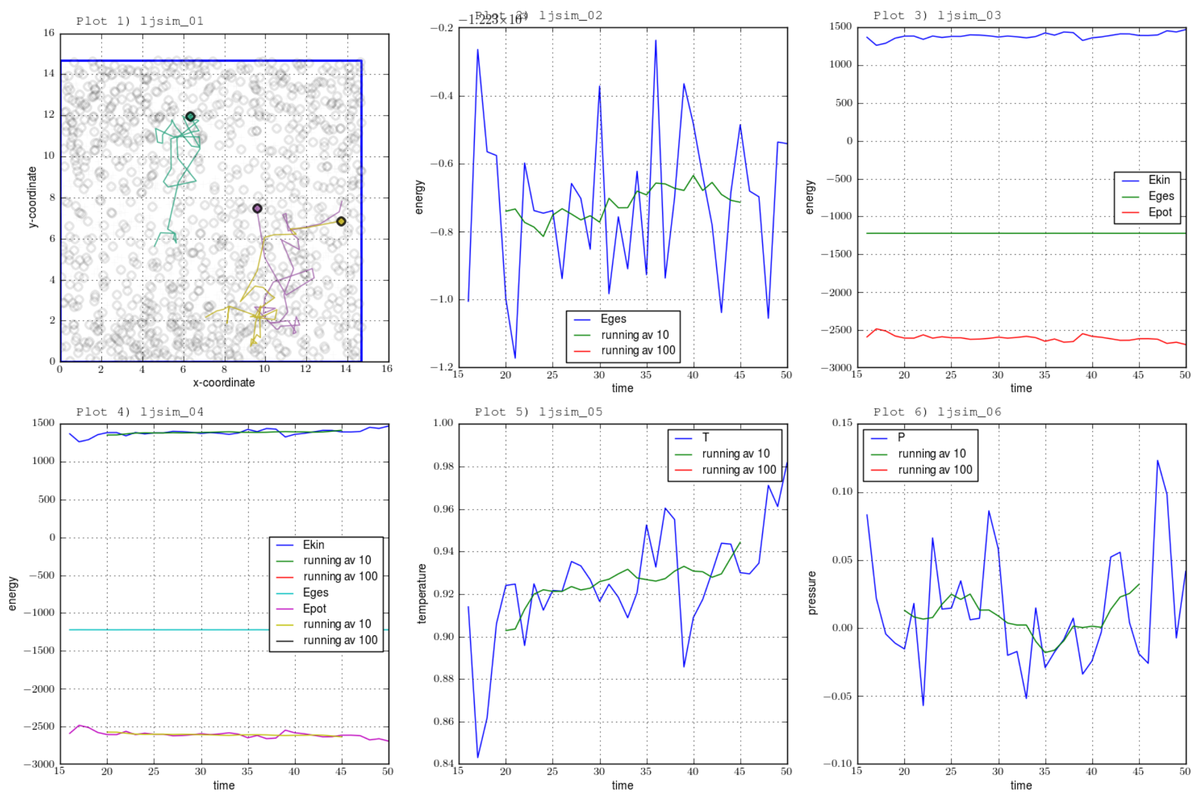


Figure 2: Observables up to t=1000 and their running averages

In this context it should be mentioned, that there is no running average for window size 100 as we only simulated 50 time steps. The mean values of the observables measured in one of our simulatios are:

Observable	Mean value
E_{tot}	1329.25
E_{pot}	-1671.12
E_{kin}	3000.38
T	2.00
P	0.6

3 Velocity rescaling

3.1 Derivation of the rescaling factor

It is:

$$\frac{k_bT}{2} = \frac{0.5mv^2}{3N} \tag{1}$$

Which obviously leads to:

$$T_{mes} = \frac{mv_{mes}^2}{3Nk_B}$$

$$T_{des} = \frac{mv_{des}^2}{3Nk_B}$$

where T_{mes} is the measured and T_{des} is the desired temperature - analogous for the velocities. Calculating the factor $\frac{T_{mes}}{T_{des}}$ and solving for v_{des} leads to:

$$v_{des} = v_{mes}\sqrt{\frac{T_{des}}{T_{mes}}} \tag{2}$$

Therefore you have to multiply the measured velocities v_{mes} by the factor $\sqrt{\frac{T_{des}}{T_{mes}}}$ in order to get a correct velocity rescaling.

This velocity rescaling has been implemented in the python part due to the fact that it's a simple multiplication with a factor and nothing numerically problematic. Implementing it in C had only made this calculation more komlpex to implement and wouldn't bring a big simulation time reduction. The plots of all observables for $T_{des} = 0.3$, 1.0 and 2.0 are the following:

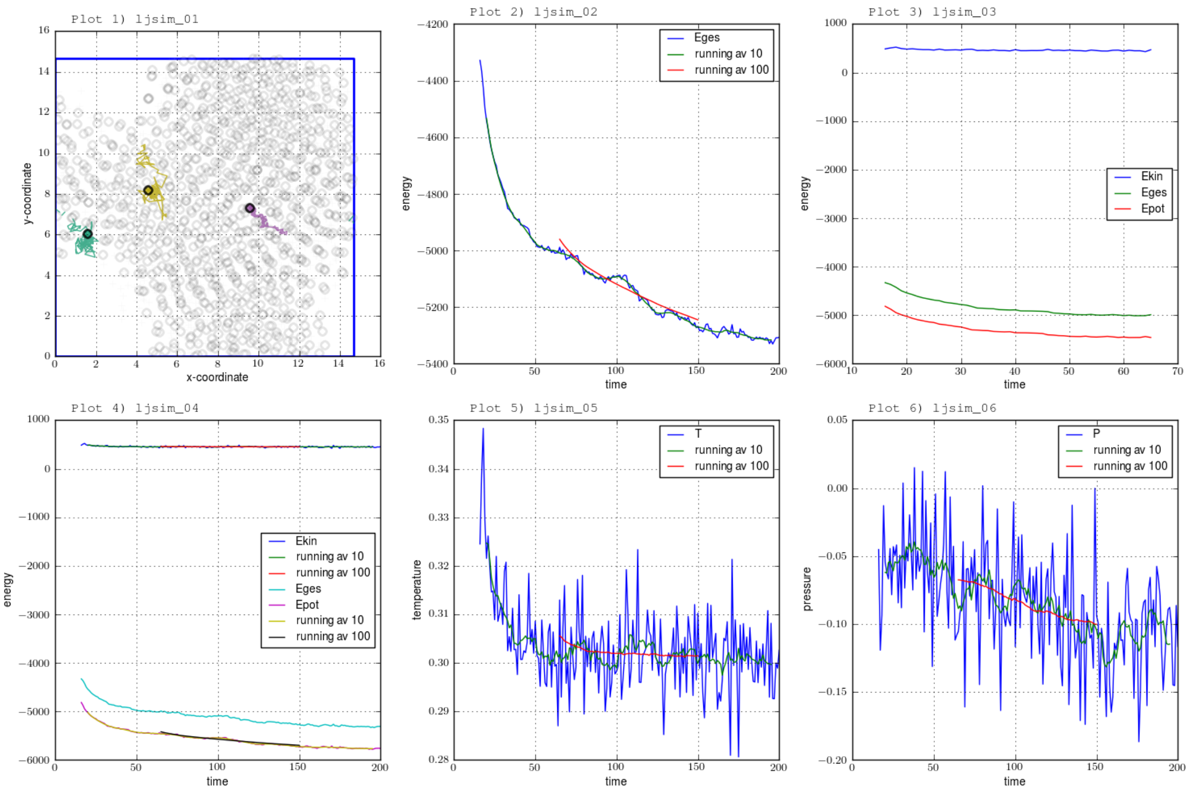


Figure 3: Observables for $T_{des} = 0.3$ and their running averages

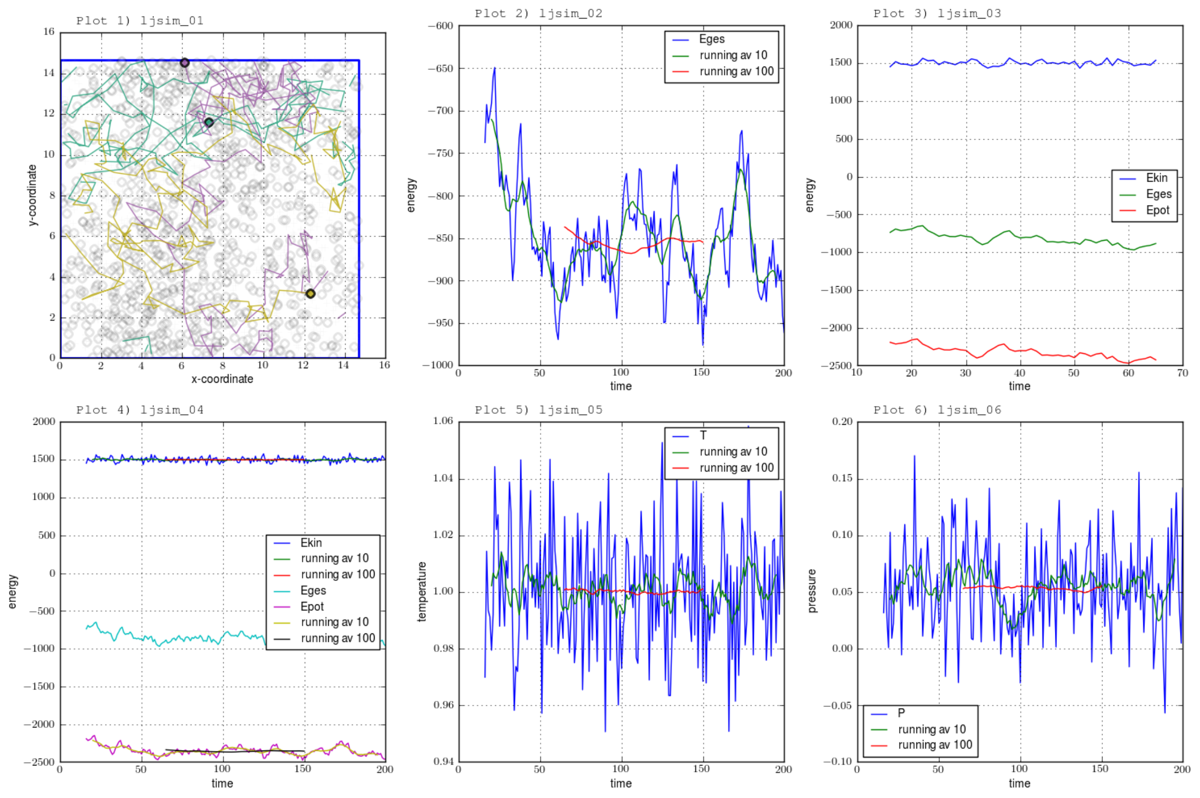


Figure 4: Observables for $T_{des} = 1.0$ and their running averages

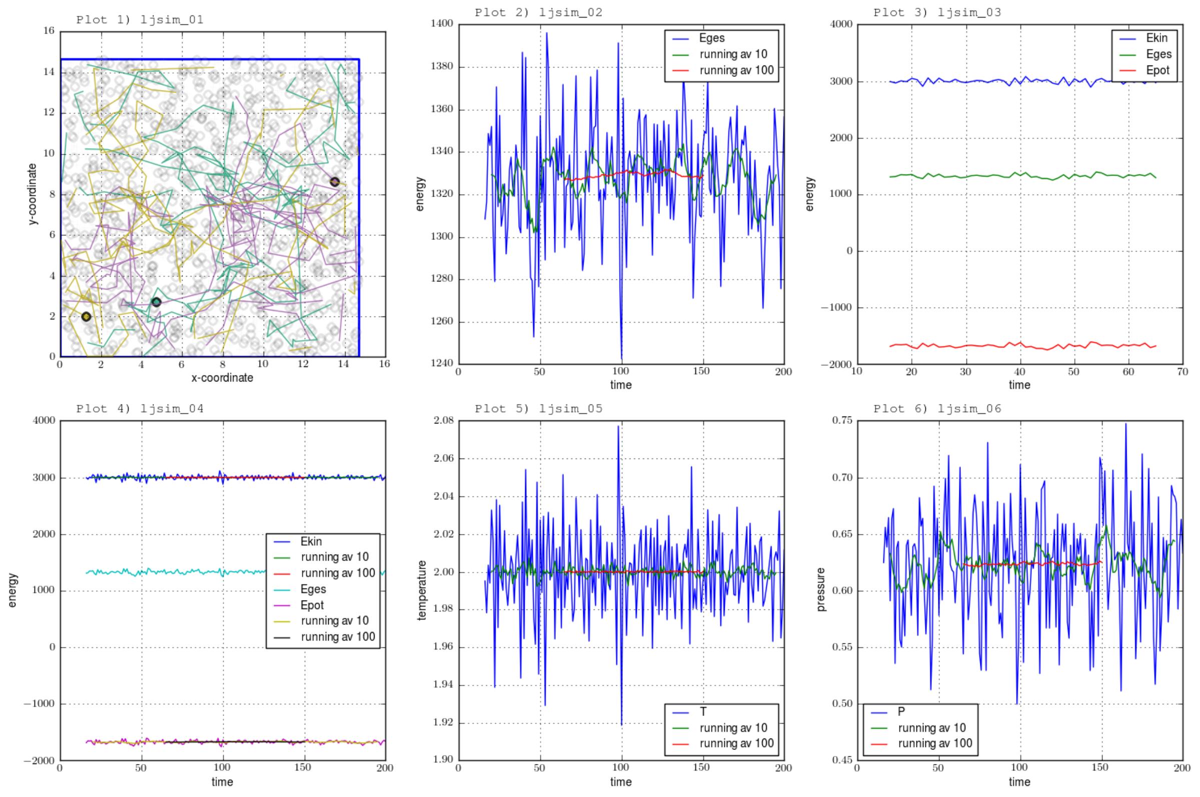


Figure 5: Observables for $T_{des} = 2.0$ and their running averages

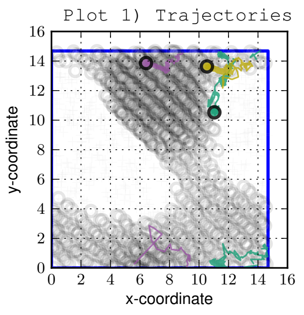
Comparing these three plots it can easily be seen that the total energies increase with the temperature which would be expected, due to the fact that the kinetic energy will increase with temperature. It also seems that equilibration will take longer for lower temperatures which can be explained by the lower average kinetic energy and therefore averagely slower movement of the particles.

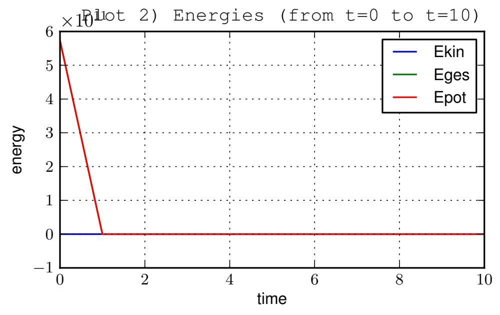
4 Force capping

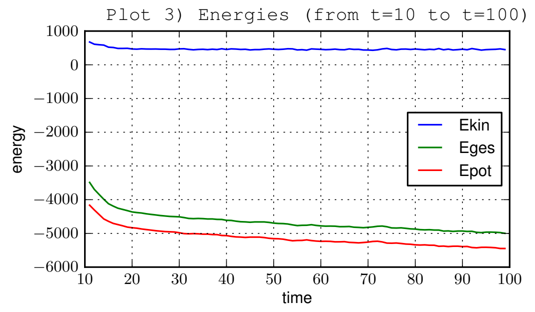
Force capping was implemented in C-part because it belongs in the function `c_compute_forces`. In addition it's a more or less complex operation when you want to do it the right way (the simple way would by to do force capping component by component).
It's important to rescale the velocities regularly when you use force capping. In addition the initially calculated observables contain large errors so that you have to reject them for analyzing. If you give attention to both of these aspects, the new set up will work well.

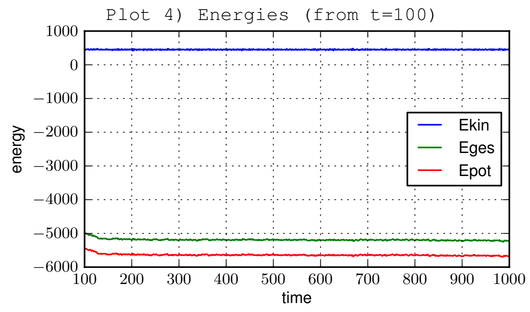
5 Plots with RDF

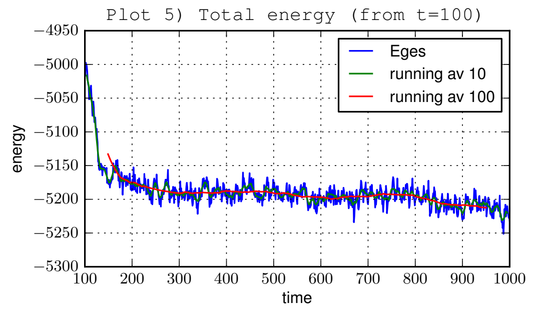
For T=0.3

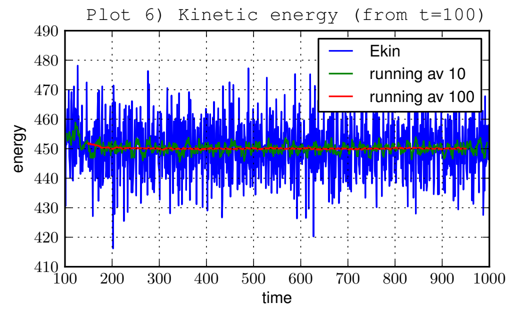


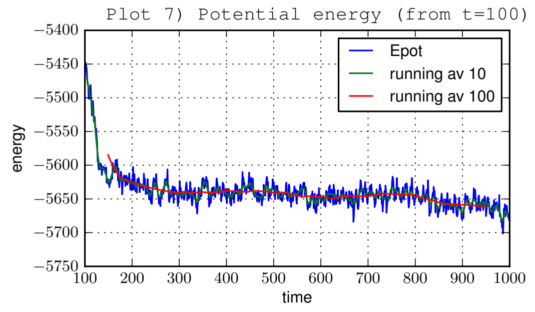


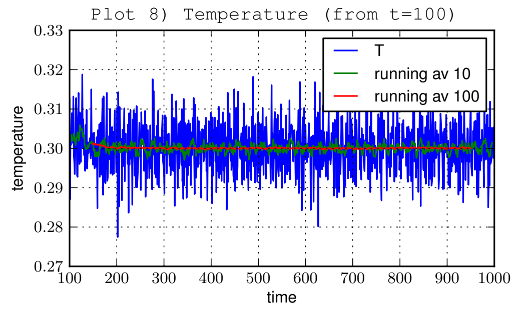


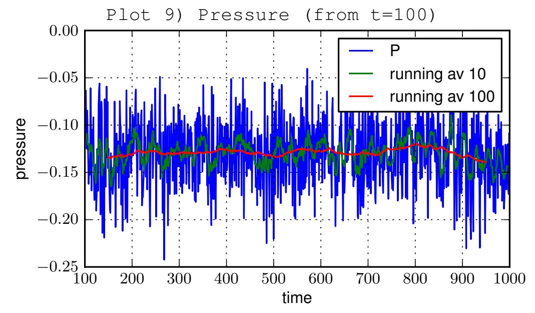


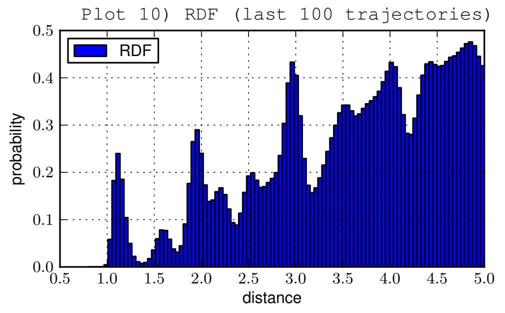




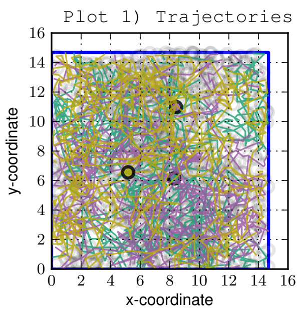


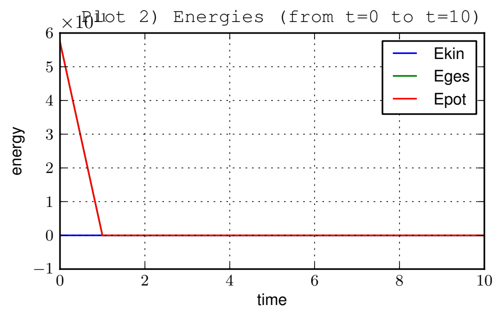


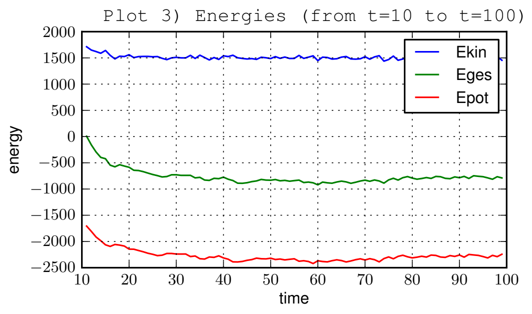


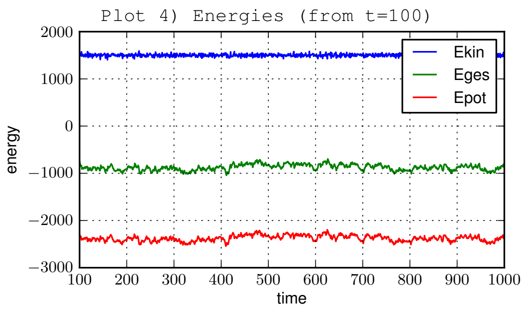


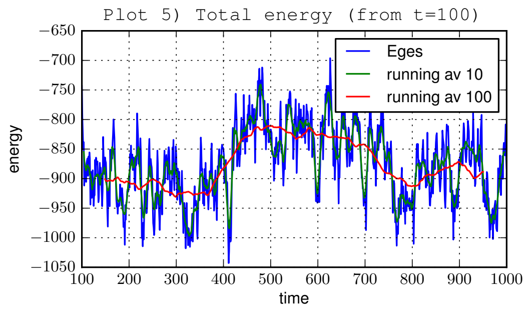
For T=1.0

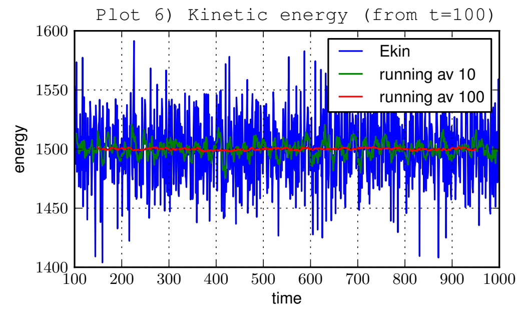

Plot 1) Trajectories
y-coordinate
x-coordinate

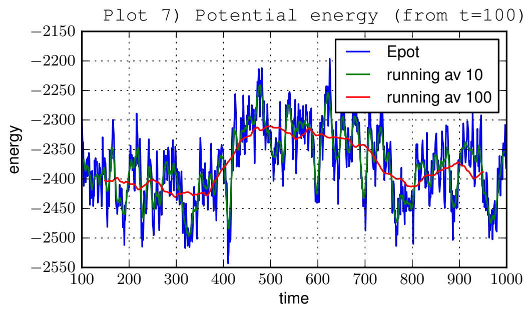

Plot 2) Energies (from t=0 to t=10)
×10
energy
time
Ekin
Eges
Epot

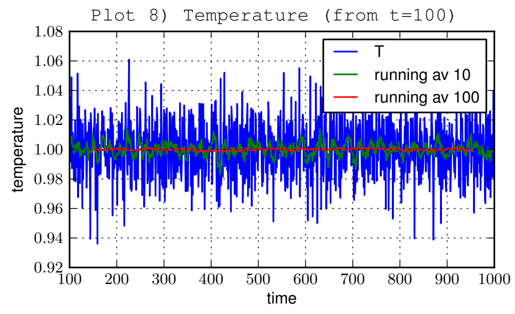

Plot 3) Energies (from t=10 to t=100)
energy
time
Ekin
Eges
Epot

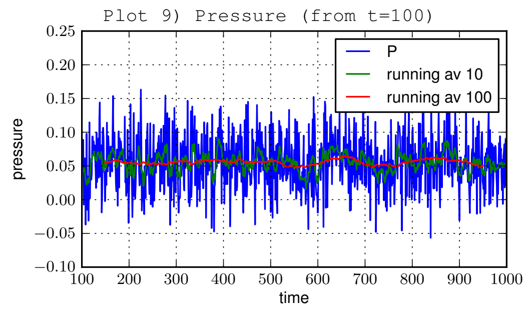

Plot 4) Energies (from t=100)
energy
time
Ekin
Eges
Epot

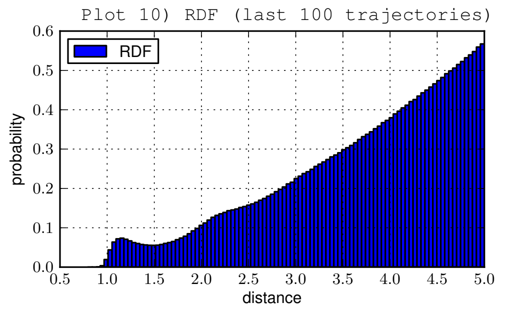

Plot 5) Total energy (from t=100)
energy
time
Eges
running av 10
running av 100


Plot 6) Kinetic energy (from t=100)
energy
time
Ekin
running av 10
running av 100

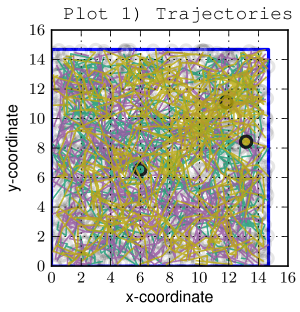

Plot 7) Potential energy (from t=100)
energy
time
Epot
running av 10
running av 100

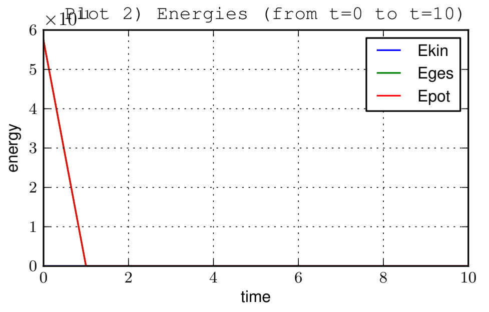

Plot 8) Temperature (from t=100)
temperature
time
T
running av 10
running av 100

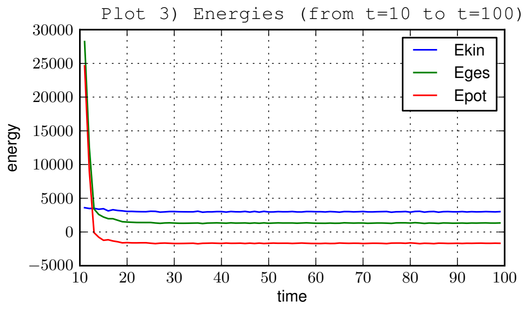

Plot 9) Pressure (from t=100)
pressure
time
P
running av 10
running av 100

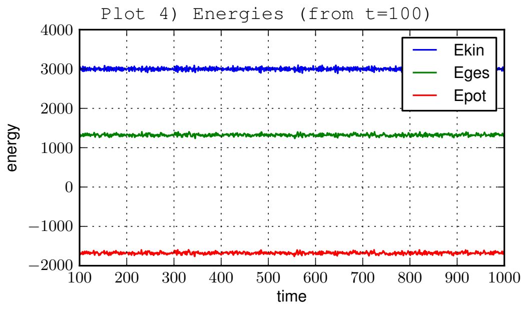

Plot 10) RDF (last 100 trajectories)
probability
distance
RDF

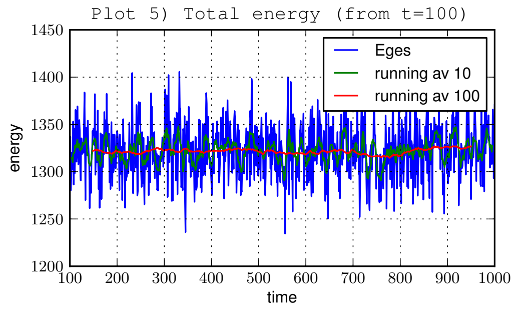
For T=2.0

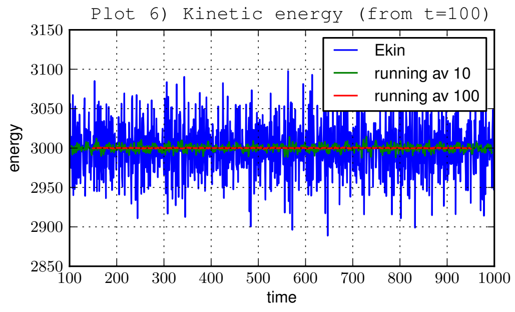

Plot 1) Trajectories
y-coordinate
x-coordinate

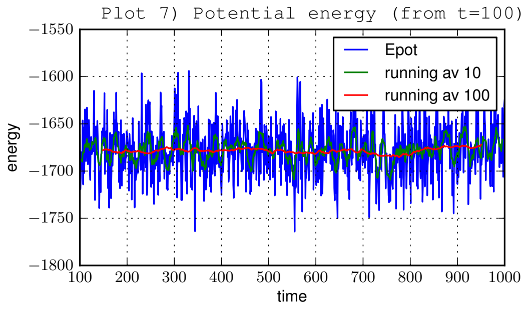

Plot 2) Energies (from t=0 to t=10)
×10
energy
time
Ekin
Eges
Epot

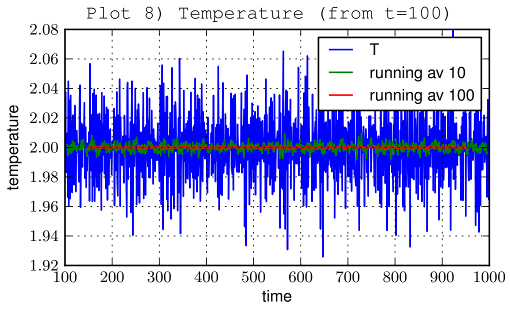

Plot 3) Energies (from t=10 to t=100)
energy
time
Ekin
Eges
Epot

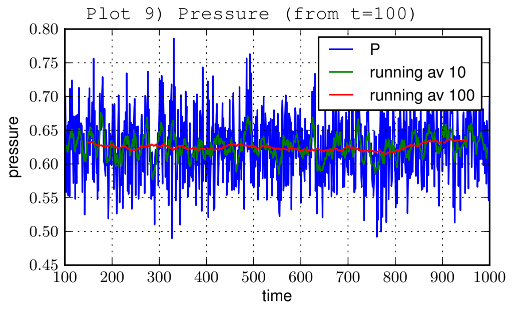

Plot 4) Energies (from t=100)
energy
time
Ekin
Eges
Epot

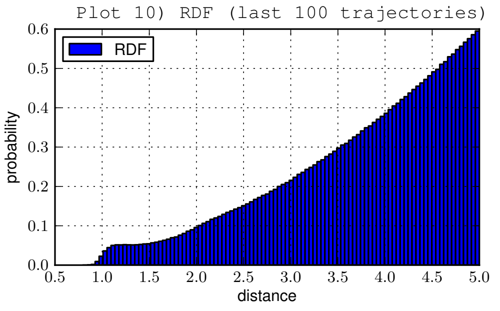

Plot 5) Total energy (from t=100)
energy
time
Eges
running av 10
running av 100


Plot 6) Kinetic energy (from t=100)
energy
time
Ekin
running av 10
running av 100


Plot 7) Potential energy (from t=100)
energy
time
Epot
running av 10
running av 100


Plot 8) Temperature (from t=100)
temperature
time
T
running av 10
running av 100


Plot 9) Pressure (from t=100)
pressure
time
P
running av 10
running av 100


Plot 10) RDF (last 100 trajectories)
probability
distance
RDF