Solutions - Exercise sheet 1

Statistical mechanics and Monte Carlo algorithm basics

PUE Advanced Computational Physics

University of Vienna - Faculty of Physics

1 Reduced units

a. Write down formulas for the reduced units of energy (E^*), pressure (p^*), time (t^*), viscosity (η^*), action (S^*), and charge (q^*).

In the LJ system of units, the base units are mass m, particle radius σ, and potential depth ε. Reduced quantities are obtained by multiplying the quantity with an appropriate combination of these base units, as well as natural constants like k_B if needed. See the separate table for all results.

The case is a bit more complicated for charge. First, depending on which system of units one chooses, not only the unit, but also the *dimension* of charge is different. For example, in its most general form, Coulomb's law for the magnitude of force between two charges q_1 and q_2, separated by a distance r, is

$$F = k_e \frac{q_1 q_2}{r^2}, \tag{1}$$

where k_e is the proportionality constant. The corresponding Maxwell equation is

$$\nabla \cdot \mathbf{E} = 4\pi k_e \rho, \tag{2}$$

where $\mathbf{E}$ is the electric field and ρ is the charge density. Many will be familiar with the SI choice, $k_e^\mathrm{SI} = 1/4\pi\varepsilon_0$, with the vacuum permittivity $\varepsilon_0 = 8.854\,187\,817... \times 10^{-12}\,\mathrm{F/m}$ (farads per meter). In SI, the unit of charge is C (coulomb), which is the same as As (ampere second). In other words, the unit of charge is derived by multiplying two base units. In (Gaussian) cgs units, we have $k_e = 1$, so there is no explicit constant of proportionality in Coulomb's law. Conversely, the dimension of charge has to change, because ultimately, there still has to be a force on the L.H.S. of Eq. (1). In cgs, the unit of charge is Fr (franklin), which is nothing but $\mathrm{cm}^{3/2}\,\mathrm{g}^{1/2}\,\mathrm{s}^{-1}$, so here we have a combination of three base units as unit of charge. Plug everything in and you will see that indeed the resulting quantity in Coulomb's law has units of $\mathrm{g\,cm\,s}^{-2}$ (dyn, which stands for dyne), the correct unit of force in this system.

Let us now consider an LJ system with additional charges on the particles. The potential energy between two charged particles i and j is given as

$$u(r_{ij}) = 4\varepsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] + k_e \frac{q_1 q_2}{r_{ij}}. \tag{3}$$

For consistency, the Coulomb term needs to have the same units as the LJ term, which means it has to have units of ε. Conversely, this requirement determines the unit of charge. The result is $q^* = q/\sqrt{\sigma\varepsilon/k_e}$. In other words, also in LJ units, the unit of charge is derived from the base units (and a natural constant) and cannot be set independently. In the following, for

simplicity I will assume $k_e = 1$. What if you want to use the elementary charge q_e as a base unit? You can certainly do that, but it means that you cannot set σ and ε independent of each other any more, because the relation $q_\mathrm{e} = \sqrt{\sigma\varepsilon}$ has to hold. To see this intuitively, think about two charges of $1\,q_\mathrm{e}$ each, separated by a distance of $1\,\sigma$. Once you set for example $\sigma = 3.405\,\text{Å}$, the Coulomb potential energy value will also be fixed—effectively determining the physical value of ε. This also means that it is not possible to use q_e as your charge unit, but leave σ and ε in their "abstract" or undefined form. In practice, q_e is very convenient to use, but you will have to set k_e accordingly. For example, using LAMMPS' "real" units, the value is $k_e = 332.06371\,\text{kcal/mol Å}\,q_\mathrm{e}^{-2}$.

b. Using the LJ parameters of argon, which pressure corresponds to $p^* = 1$? A typical timestep used in molecular dynamics simulations is $t^* = 0.005$. Which timestep does this correspond to?

We have

$$p = \frac{\varepsilon p^*}{\sigma^3} \tag{4}$$

and set $p^* = 1$.

$$\begin{aligned} p &= 119.8\,\text{K} \cdot k_\mathrm{B} / (3.405\,\text{Å})^3 \\ &= 119.8 \times 1.38 \times 10^7\ \text{kg m}^{-1}\,\text{s}^{-2} \,/\, 3.405^3 \\ &= 41,897,547.4\,\text{Pa} \\ &= 418.975\,\text{bar}. \end{aligned}$$

For time, the relation is

$$t = t^* \sigma \sqrt{m/\varepsilon}. \tag{5}$$

We set $t^* = 0.005$ and use the mass of an argon atom $m = 39.792$ amu (atomic mass units), where 1 amu = 1 g/mol = 1 g/N_A = 1.66×10^{-27} kg. Strictly speaking, the mole is an SI base unit while N_A is a pure number, so these equations are a bit sloppy in terms of dimensions. However, in my experience the best way of dealing with moles is to get rid of them ASAP. The final result is

$$\begin{aligned} \Delta t &= 1.076 \times 10^{-14}\,\text{s} \\ &= 10.76\,\text{fs}. \end{aligned}$$

c. Based on the "real"-units of length, time, and mass, what are the expected units of force and energy? Compare this to the unit of energy that is used by LAMMPS; what is the conversion factor between the two units?

- mass = g/mol = 1.66×10^{-27} kg
- length = Å
- time = fs

The unit of energy derived from these base units is g/mol $\text{Å}^2\,\text{fs}^{-2}$. However, LAMMPS uses kcal/mol, a common unit in chemistry and related fields. We solve for the conversion factor x

between these two units:

$$\begin{aligned}
\text{kcal/mol} &= x\ \text{g/mol}\,\text{Å}^2\,\text{fs}^{-2} \\
\text{kcal} &= x\ \text{g}\,\text{Å}^2\,\text{fs}^{-2} \\
4184\,\text{J} &= x\ \text{g}\,\text{Å}^2\,\text{fs}^{-2} \\
4184\,\text{kg}\,\text{m}^2\,\text{s}^{-2} &= x\ 10^7\,\text{kg}\,\text{m}^2\,\text{s}^{-2} \\
4.184 \times 10^{-4} &= x.
\end{aligned}$$

The inverse is $1/x = 2390.057361376673$, which can be found in LAMMPS' source code (file: `update.cpp`) in the form `force->mvv2e = 48.88821291 * 48.88821291`. As a side remark, coming back to charges, in the same file, you will also find `force->qqr2e = 332.06371`.

d. What is the numerical value that corresponds to a pressure of 1 atm in these units?

The "natural" unit of pressure in these units is $\text{g/mol}\,\text{Å}^{-1}\,\text{fs}^{-2}$. This corresponds with 1.66×10^{13} Pa, or 1.66×10^8 bar. Since 1 atm = 1.01325 bar, this corresponds with $p \approx 6.1 \times 10^{-9}\ \text{g/mol}\,\text{Å}^{-1}\,\text{fs}^{-2}$.

e. What is the value of $\hbar$ in these units?

We apply the same strategy as in problem c to convert from SI units to (expected) LAMMPS units. SI in, the unit of $\hbar$ is Js, or $\text{kg}\,\text{m}^2\,\text{s}^{-1}$.

$$\begin{aligned}
\text{kg}\,\text{m}^2\,\text{s}^{-1} &= x\ \text{g/mol}\,\text{Å}^2\,\text{fs}^{-1} \\
&= x\ 1.66 \times 10^{-32}\,\text{kg}\,\text{m}^2\,\text{s}^{-1}. \\
x &= 6.022 \times 10^{31}.
\end{aligned}$$

Using $\hbar = 1.054\,57 \times 10^{-34}$ Js, the result is $\hbar = 0.00635\ \text{g/mol}\,\text{Å}^2\,\text{fs}^{-1}$. Similarly, with kcal/mol as energy unit, we get $\hbar = 15.172$ kcal/mol fs, or $h = 95.328$ kcal/mol fs. Again, this can be found in the source code in the form `force->hplanck = 95.306976368`. Also note that as expected the conversion factor between the two variants of expressing $\hbar$ is the same as the one calculated in problem c.

As a last remark, the problem of units is handled differently by different simulations codes. In LAMMPS, all calculations are done using the selected system of units, which necessitates that proportionality constants like k_e or k_B are set to their respective values. CP2K (a package with a focus on electronic structure calculations—https://www.cp2k.org/) on the other hand does everything in a fixed system of internal units and applies conversion factors only for input and output.

2 A basic LJ MC Simulation

See the attached Jupyter notebook for all results.

[1]:
```
import numpy as np
from scipy import constants
import matplotlib.pyplot as plt
```

[2]:
```
eps_over_kB = 119.8 # K
sigma = 3.405 # Angstroms
N = 256
T = 83.8 # K
V = 28.24 # cm^3 / mol
```

[3]:
```
T_star = T / eps_over_kB

rho_star = 1/( (V/constants.N_A) / (1e-8 * sigma)**3)

print("T* = {}".format(T_star))
print("rho* = {}".format(rho_star))
```

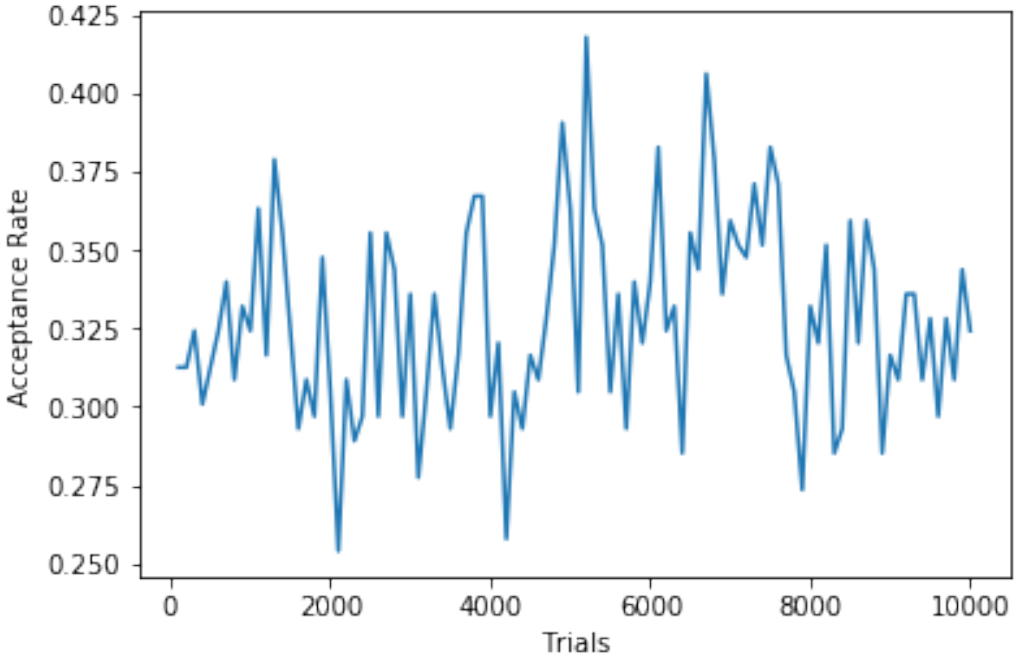
```
T* = 0.6994991652754591
rho* = 0.8418555100477527
```

1 Equilibration

[4]:
```
equ_data = np.loadtxt("equilibration/eq_log.txt")
```

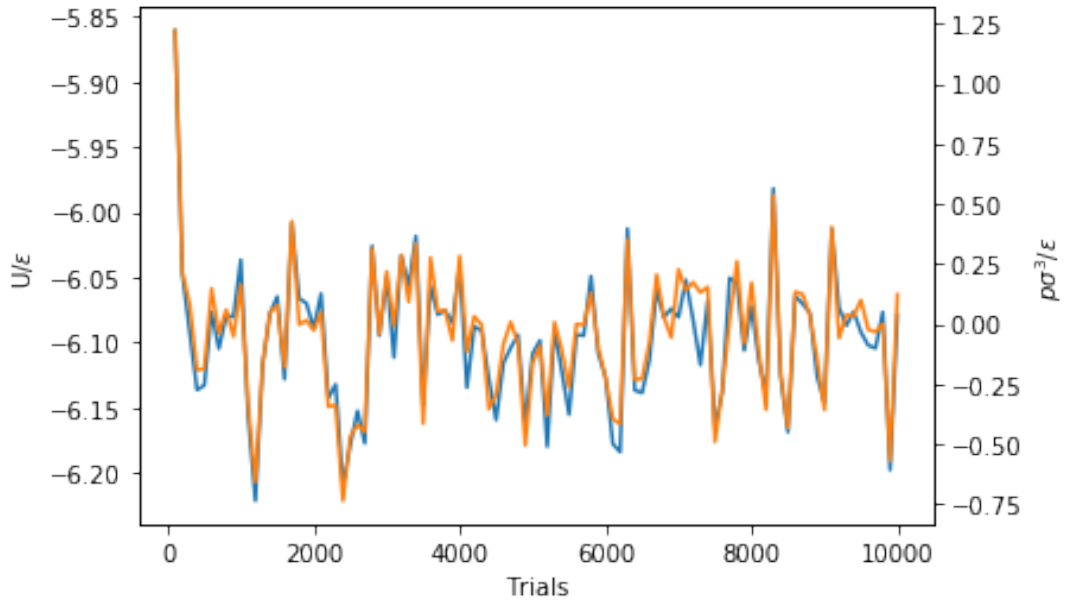
[5]:
```
plt.plot(equ_data[:,0], equ_data[:,1])
plt.xlabel("Trials")
plt.ylabel("Acceptance Rate")
```

[5]:
```
Text(0, 0.5, 'Acceptance Rate')
```



[6]:
```
plt.plot(equ_data[:,0], equ_data[:,2])
plt.xlabel("Trials")
plt.ylabel("U/$\\epsilon$")
plt.twinx()
plt.plot(equ_data[:,0], equ_data[:,3], color='C1')
plt.xlabel("Trials")
plt.ylabel("$p\\sigma^3$/$\\epsilon$")
```

[6]:
```
Text(0, 0.5, '$p\\sigma^3$/$\\epsilon$')
```

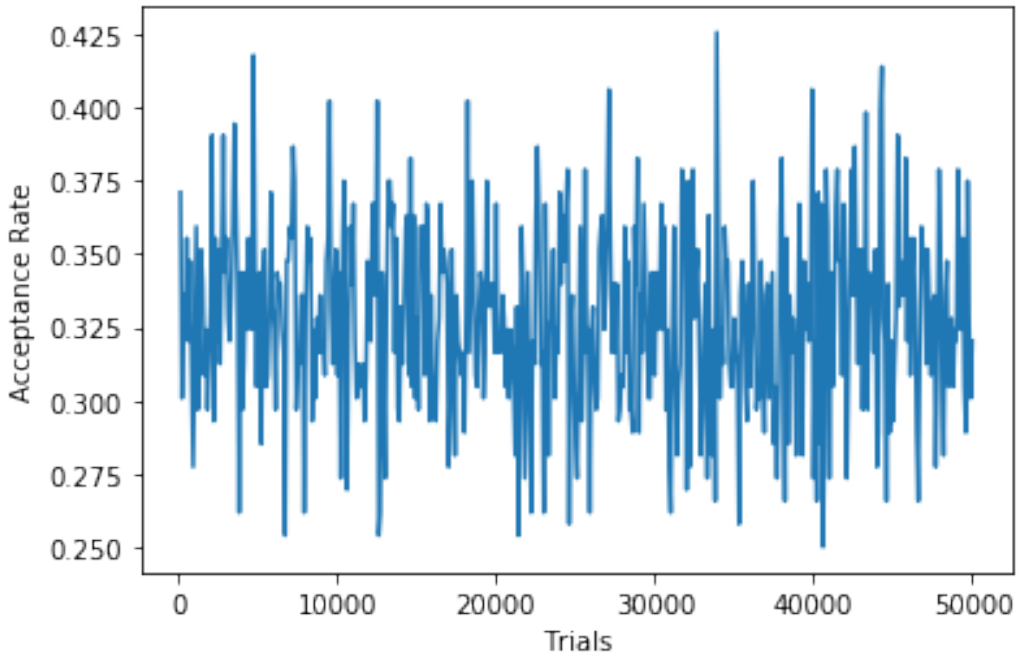


2 Production

[7]:
```
prod_data = np.loadtxt("production/prod_log.txt")
```

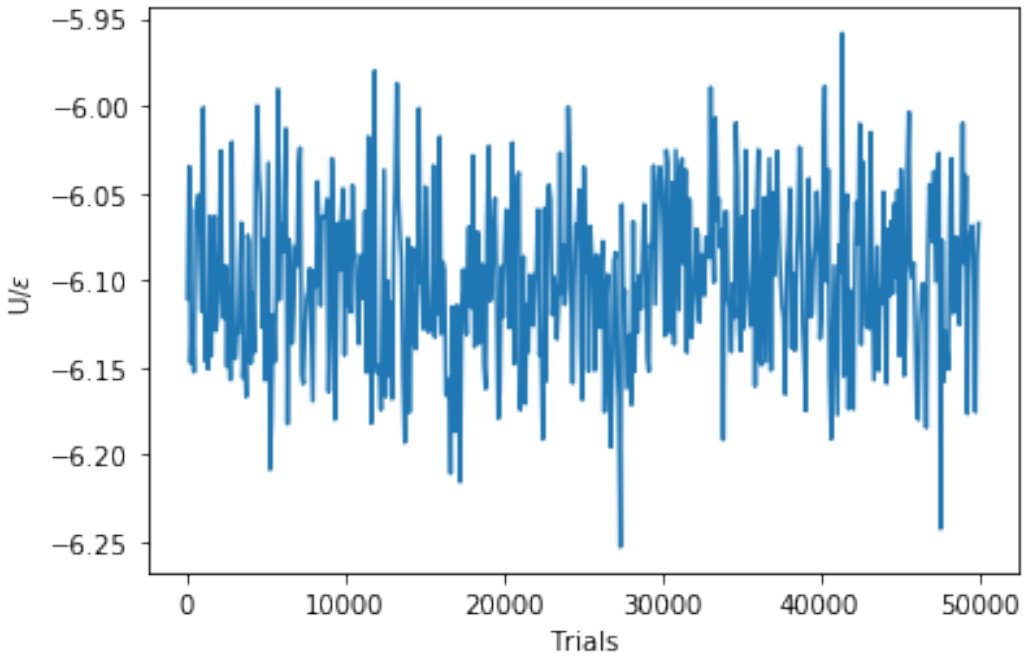
[8]:
```
plt.plot(prod_data[:,0], prod_data[:,1])
plt.xlabel("Trials")
plt.ylabel("Acceptance Rate")
```

[8]:
```
Text(0, 0.5, 'Acceptance Rate')
```



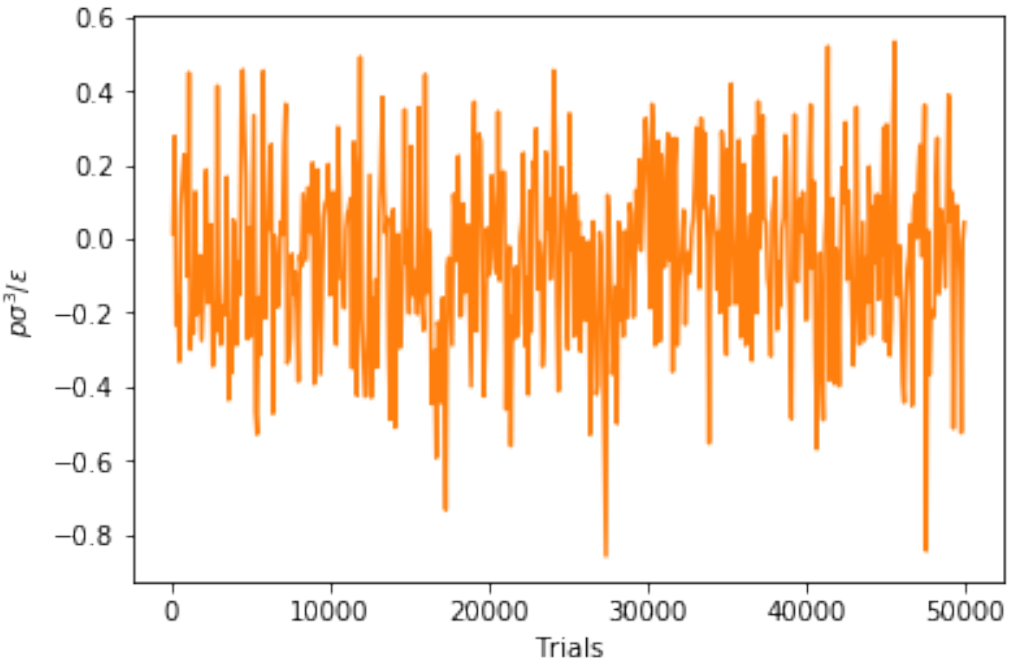
[9]:
```
plt.plot(prod_data[:,0], prod_data[:,2])
plt.xlabel("Trials")
plt.ylabel("U/$\\epsilon$")
```

[9]: `Text(0, 0.5, 'U/$\\epsilon$')`



[10]:
```
plt.plot(prod_data[:,0], prod_data[:,3], color='C1')
plt.xlabel("Trials")
plt.ylabel("$p\\sigma^3$/$\\epsilon$")
```

[10]:
```
Text(0, 0.5, '$p\\sigma^3$/$\\epsilon$')
```



[11]:
```
avg_U_star = np.average(prod_data[:,2])
avg_U = avg_U_star * eps_over_kB * constants.k * constants.N_A / 1000

print("<U*> = {}".format(avg_U_star))
print("<U> = {}".format(avg_U))

std_U_star = np.std(prod_data[:,2])
std_U = std_U_star * eps_over_kB * constants.k * constants.N_A / 1000

print("std(U*) = {}".format(std_U_star))
print("std(U) = {} kJ/mol".format(std_U))
```

```
<U*> = -6.09902542
<U> = -6.075072239638414
std(U*) = 0.04691370208397115
std(U) = 0.04672945422631144 kJ/mol
```

[12]:
```
avg_p_star = np.average(prod_data[:,3])
avg_p = avg_p_star * eps_over_kB * constants.k /( 1e-10*sigma)**3 / 1e5

print("<p*> = {}".format(avg_p_star))
print("<p> = {} bar".format(avg_p))

std_p_star = np.std(prod_data[:,3])
std_p = std_p_star * eps_over_kB * constants.k /( 1e-10*sigma)**3 / 1e5

print("std(p*) = {}".format(std_p_star))
print("std(p) = {} bar".format(std_p))
```

```
<p*> = -0.051067115588000006
<p> = -21.395876399386072 bar
std(p*) = 0.23371999630175552
std(p) = 97.92298028503508 bar
```

[16]:
```
C_V_star = np.var(prod_data[:, 2] * N) / T_star**2 / N
C_V = C_V_star * constants.k*constants.N_A
print("C_V_ex* = {} ".format(C_V_star))
print("C_V_ex = {} J/(K mol)".format(C_V))
print("\nC_V_id* = {}".format(1.5))
print("C_V_id = {} J/(K mol)".format(1.5*constants.k*constants.N_A))

print("\nC_V_id* + C_V_ex* = {} ".format(1.5 + C_V_star))
print("C_V_id + C_V_ex = {} J/(K mol)".format((1.5 + C_V_star) *constants.
 ↪k*constants.N_A))
```

```
C_V_ex* = 1.1515027391992645
C_V_ex = 9.574126479773344 J/(K mol)

C_V_id* = 1.5
C_V_id = 12.471693927229861 J/(K mol)

C_V_id* + C_V_ex* = 2.6515027391992643
C_V_id + C_V_ex = 22.045820407003202 J/(K mol)
```