Molecular Dynamics and Monte Carlo simulation of Lennard-Jones systems

-a tutorial-

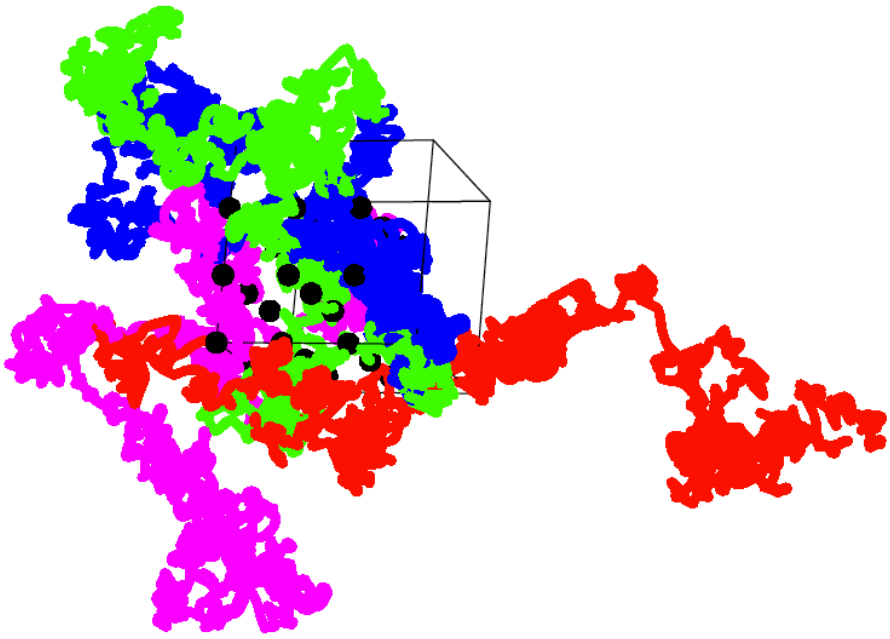
Reinhard Hentschke*

School of Mathematics and Natural Sciences

Bergische Universität, D-42097 Wuppertal, Germany

-March, 2019-

E-mail: hentschk@uni-wuppertal.de



Abstract

This is a brief introduction to Molecular Dynamics and Metropolis Monte Carlo simulation techniques and their application to simple molecular fluids. The background material, in particular Statistical Mechanics, is reduced to a minimum. A number of small exercises and two more extensive simulation projects, thermal conductivity and gas-liquid phase coexistence, teach the reader how to apply molecular simulations and how to compare the results to experimental measurements.
In the current version of this tutorial I have modified the thermal conductivity problem in response to feedback from several students.

Technical preliminaries

Simulation boxes

Imagine 18g of water - roughly 2.6 cm $\times$ 2.6 cm $\times$ 2.6 cm in terms of volume. How many molecules does this much water contain? About $6 \cdot 10^{23}$. There is no computer yet that can handle this many molecules. We in fact will deal with somewhere between 100 to 300 molecules. Nevertheless, we have to trick them into 'thinking' that they are $6 \cdot 10^{23}$. This big a system is a bulk system - a system in which the surfaces do not affect the properties.

The red particles in Fig. 1 are models of real molecules stored in a computer's memory. Every particle is the center of a circle with radius r_{cut}. Things outside its circle a particle does not 'see' directly. This means that two particles interact directly only if they are inside each others circles.

The blue particles in Fig. 1 are not real molecules. They are not stored anywhere on the computer. They have well defined positions, however, because they are periodic images of the red particles. The periodicity arises because of the central square. The latter defines a lattice and every lattice cell contains exactly the same particles at the exact same positions. In some cases the particles are real (red), but in most cases

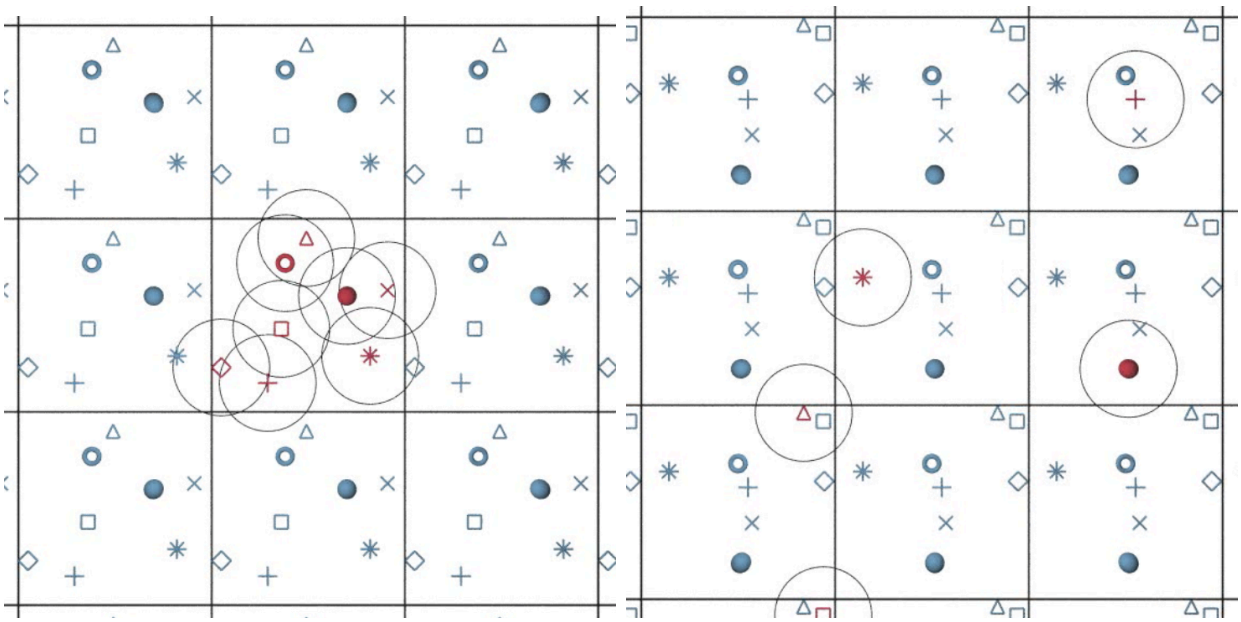


Figure 1: Left: Primary simulation box in the center at the beginning of a simulation embedded in a lattice of its periodic images. Right: After a certain number of simulation steps all but one of the real, i.e. red, particles have left the original simulation box. The original density, if the image, i.e. blue, particles are included in the count, has not changed.

the are merely images of real particles (blue). The volume of the central square, or (primary) simulation box, determines the (number) density of particles, i.e. molecules. As long as the range of interaction (circle) is less, ideally much less, than the size of the simulation box, a particle should not notice the small size of its system. In practice this is not exactly true and we must expect so called *finite size effects*.

When we calculate the path of a red particle in space, we do this based on all interactions with particles inside its interaction radius, r_{cut}, which we call cutoff radius. These particles can be red or blue! In the right panel of Fig. 1 the last red particle in the central box interacts with only the blue diamond. The real red diamond, however, is a long way off. Since we only store the position of real, i.e. red, particles, we must use the position of the red diamond to determine the distance between the red star in the central box and the blue diamond in its cutoff radius. This is done, here for the x-distance, by the following line of computer code:

$$x_{ij}^{min} = x_{ij} - L \,\text{Round}[x_{ij}/L] \tag{1}$$

The quantity x_{ij} is given by $x_{ij} = x_i - x_j$, where x_i is the x- coordinate of the red star and x_j is the x-coordinate of the red diamond. L is the length of the primary simulation cell in x-direction. Round$[a]$ returns a rounded to the nearest integer. No matter where the red diamond really is, the magnitude of x_{ij}^{min} is the x-distance separating the red star from the nearest image of the red diamond. Thus, when we calculate the potential energy of a real particle i in a system, we do this as follows. First we find every particle, j ($\neq i$), the real j or one of its images, inside the cutoff radius of i using Eq. (1). That is the condition $r_{cut} > r_{ij}^{min}$, with

$$(r_{ij}^{min})^2 = (x_{ij}^{min})^2 + (y_{ij}^{min})^2 + (z_{ij}^{min})^2 \,, \tag{2}$$

must be satisfied. Subsequently we compute the pair-potential energies based on these distances r_{ij}^{min}. Likewise we proceed in the case of forces. Eq. (1) is also known as *minimum image convention*. Using the minimum image convention the real particles are free to move wherever they like, while the density remains constant. The simulation box mimics a bulk system. One last note. Some simulations require that the density is variable, i.e. the pressure is constant instead of the volume. We can handle constant pressure by varying L and still use (1) as described.

Error calculation

There are many types of errors. Some errors are simply mistakes. Others are due to numerical inaccuracies of the simulation algorithm. Still others have to do with insufficient equilibration. Here I want to talk about statistical errors.

Every simulation algorithm produces long sequence of numbers. One such sequence

may be the x-position of a particle, another one the potential energy or the temperature. Whatever it is, we call this quantity A and its values A_i. The sample average of A is

$$\bar{A} = \frac{1}{K} \sum_{i=1}^{K} A_i \tag{3}$$

and

$$s_A^2 = \bar{A^2} - \bar{A}^2 \; . \tag{4}$$

is the sample variance. Fig. 2 depicts a mock series of data points. After an equilibration phase, which may look different from the more or less monotonous increase shown here, the data points finally form a 'plateau'. Only the data in this plateau are used for analysis.

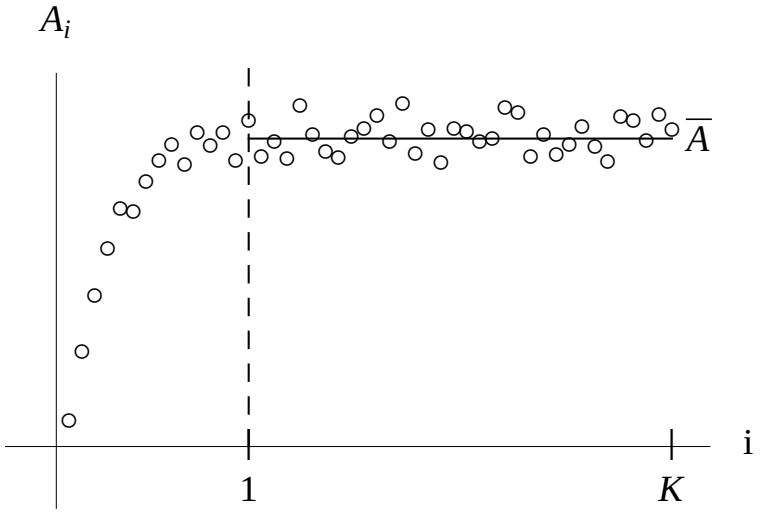


Figure 2: Mock series of data points A_i including an initial equilibration phase followed by equilibrium data. The dashed line separates the data aquired during equilibration from the data used for analysis. The horizontal line is the equilibrium average $\bar{A}$.

Using the *central limit theorem* one can estimate the likelihood that the true average value of A, i.e. $\langle A \rangle$, satisfies

$$\bar{A} - \frac{s_A}{\sqrt{n}} < \langle A \rangle < \bar{A} + \frac{s_A}{\sqrt{n}} \; . \tag{5}$$

This likelihood is 68 %. Notice that n is the number of independent values A_i in the original sample. When you present sample averages computed from simulated data, you must include the standard error $\pm \frac{s_A}{\sqrt{n}}$. This is done either in the form of an error bar, when the sample average $\bar{A}$ is a data point in a graph, or, when $\bar{A}$ is a number, in the form $\bar{A} \pm \frac{s_A}{\sqrt{n}}$. In addition, you should refrain from presenting numerical values using more decimal places than supported by the error estimate for the respective quantity.

The number of independent values n in your series of stored simulation data can be determined from the auto-correlation function of A, i.e.

$$C_A(k) = \frac{\sum_{i=1}^{K-k}(A_i - \bar{A})(A_{i+k} - \bar{A})}{\sum_{i=1}^{K}(A_i - \bar{A})^2} \; . \tag{6}$$

Two examples for $C_A(k)$ are shown in Fig. 3. In this particular case $t = \Delta t \; k$, where $\Delta t = 0.001$ is the timestep in a Molecular Dynamics simulation (cf. below). The two curves, labeled P and T, are autocorrelation functions for pressure and temperature obtained in a simulation. Important parameters characterising the simulated system were the particle number density, $\rho = N/V = 0.15$, where V is the volume of the simulation box, the particle number, $N = 108$, the cutoff radius, $r_{cut} = 3$, and $T = 2.59$ (in Lennard-Jones units as explained below). Both auto-correlation functions have decayed to zero at $t \approx 2$, which means $k = k_c \approx 2000$. This value, i.e. $k = k_c$, beyond which $C_A(k)$ becomes zero -within small fluctuations- can be used to determine n via

$$n = K/k_c \; . \tag{7}$$

Notice that in general k_c depends on the choice of simulation parameters.

The auto-correlation function also allows to spot certain types of systematic errors, i.e. 'drifts' in the data. The inset in Fig. 3 shows C_T for two independent Molecular Dynamics simulations of different precision. The C_T obtained for the longer timestep, $\Delta t = 0.01$, does not decay to zero. Due to numerical error the temperature does not fluctuate around a constant value, which gives rise to a non-vanishing correlation.

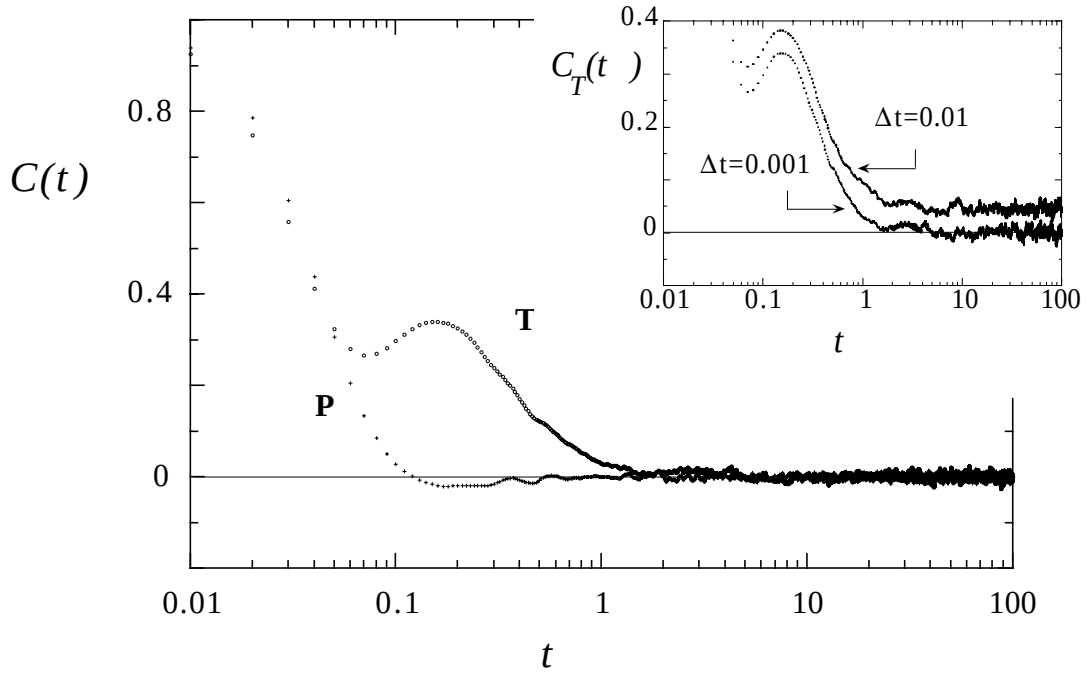


Figure 3: Examples of auto-correlation functions obtained for pressure and temperature data extracted from Molecular Dynamics simulations of a Lennard-Jones gas.

• **Exercise - standard error:** Give a justification for (5). Why is the probability that the true average is inside these limits 0.68?

• **Exercise - autocorrelation function:** Give a justification for the statement 'The value of $k = k_c$ beyond which $C_A(k)$ becomes zero -within small oscillations- can be used to determine n.'

Lennard-Jones interactions

This tutorial focusses on simple gases and liquids. 'Simple' means that the interactions between the molecules, or atoms in the case of noble gases, may be described via simple potential functions like the Lennard-Jones (LJ) potential:

$$u(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] . \tag{8}$$

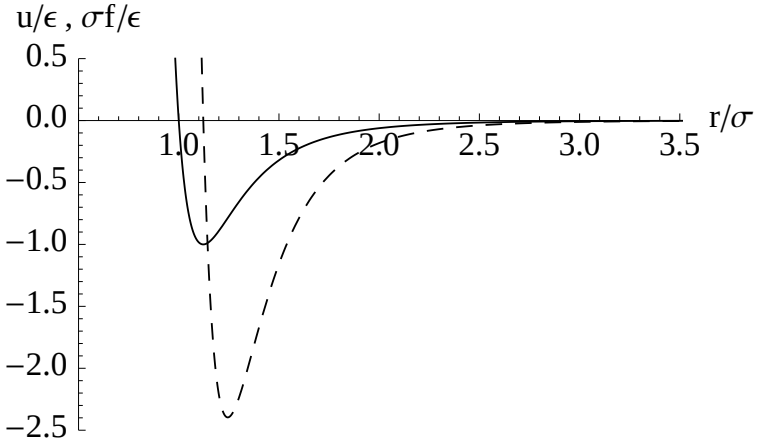


Figure 4: Lennard-Jones potential, u, (solid line) and the magnitude of its force, f, (dashed line).

Here r is the minimum image center-to-center separation of two particles, i.e. r_{ij}^{min} as explained above. When the two particles are far apart, they attract each other until their distance is less than $2^{1/6}\sigma$, where the force f is zero. For smaller separation the interaction force is repulsive. The $-r^{-6}$-attraction in the potential is the leading term of the quantum mechanical interaction between small neutral molecules, i.e. neutral molecules universally attract each other at large distances. The r^{-12}-repulsion is a convenient description of the excluded volume interaction between molecules. However, there is no deeper justification for its mathematical form.

The two parameters ϵ and σ are a typical energy and a characteristic linear dimension of the molecules, respectively. Here we deal with pure systems only, i.e. all

molecules in our simulations do have the same ϵ and σ. This makes it convenient and useful to measure all energies in units of ϵ and all lengths in units of σ. Thus, we do not use (8) as it stands but rather

$$u(r) = 4\left[r^{-12} - r^{-6}\right] \ , \tag{9}$$

i.e. your programs should never contain ϵ and/or σ explicitly!

At first glance this may seem as if we study just one very particular system. This is correct, but there is what is called *the law of corresponding states*. This law is not a strict law. It rather is an approximation - albeit a very good one for many fluids of small molecules. It means that if we have done all simulations in these units, the LJ units which we discuss in more detail below, then we can, if we know ϵ and σ for a particular molecule, map our LJ results onto the real system. In particular, this allows the comparison of the simulation results to corresponding experiments for this molecular system.

In the above section on simulation boxes we had introduced the cutoff radius, r_{cut}. Interactions of molecules beyond r_{cut} will be neglected. Looking at Fig. 4 this seems to be reasonable if for instance $r_{cut}/\sigma \geq 3$. But how much of the total (potential) energy do we actually throw away? The answer is

$$U_{lrc} = \sum_{i<j, r_{ij}^{min} > r_{cut}} u_{ij} \approx \frac{N(N-1)}{2} \frac{1}{V} \int_{r_{cut}}^{\infty} 4\pi r^2 dr u(r) \approx -\frac{8\pi}{3} \rho \sigma^3 \epsilon \left(\frac{\sigma}{r_{cut}}\right)^3 N \tag{10}$$

(cf. Ref.[3] (section IV.a)). You can understand this formula better if you set $u(r) = \epsilon$, i.e. all particle pairs have the same potential energy independent of their separation. In this case the integral is equal to V, except for a small hole with the radius r_{cut}, which cancels the factor $1/V$. The remaining number, $N(N-1)/2$, is the number of distinct pairs of particles. Finally, $u(r)(\neq \epsilon)$ accounts for the distance dependence

of the interaction between pairs. This so called *long range correction* should be small compared to the total potential energy in your system. In LJ systems $r_{cut}/\sigma = 3$ usually meets this condition.

If necessary, as for instance in Monte Carlo simulations involving moves which change the density, you can add U_{lrc} to your potential energy as a correction. However be careful. You may think that if this is always possible, you can make r_{cut} really small, e.g. $r_{cut}/\sigma \approx 1$, and thereby reduce the computational effort. This is not correct! The first $\approx$ in Eq.(10) means that we can neglect structural ordering beyond r_{cut}. Structural ordering is caused by every particle's presence, because it imposes a distance constrain on its neighbors and, depending on density, on the next-nearest neighbors as well. A pictorial illustration of these spatial correlation is shown in Fig. 5. In order for Eq.(10) to work, r_{cut} must be sufficiently large, i.e. the presence of the central particle from, from which we measure r_{cut}, does no longer influence the positions of particles beyond r_{cut}. This can be checked by computing the so called radial pair-correlation function (cf. Eq. X.14 on page 445 in Ref.[3]).

• **Exercise - long range correction:** Cutting off the interaction between particles also means setting their interaction force equal to zero. Does it make sense to also introduce a long range correction to the force on a particle? Explain your answer!

Notice that our approach amounts to approximate the interactions between particles in a simulation by *pairwise* interactions. The total interaction energy for instance is

$$U = \sum_{i<j, r_{ij}^{min} \leq r_{cut}} u_{ij} + U_{lrc} \; . \tag{11}$$

Likewise the total force on any particle i is given by

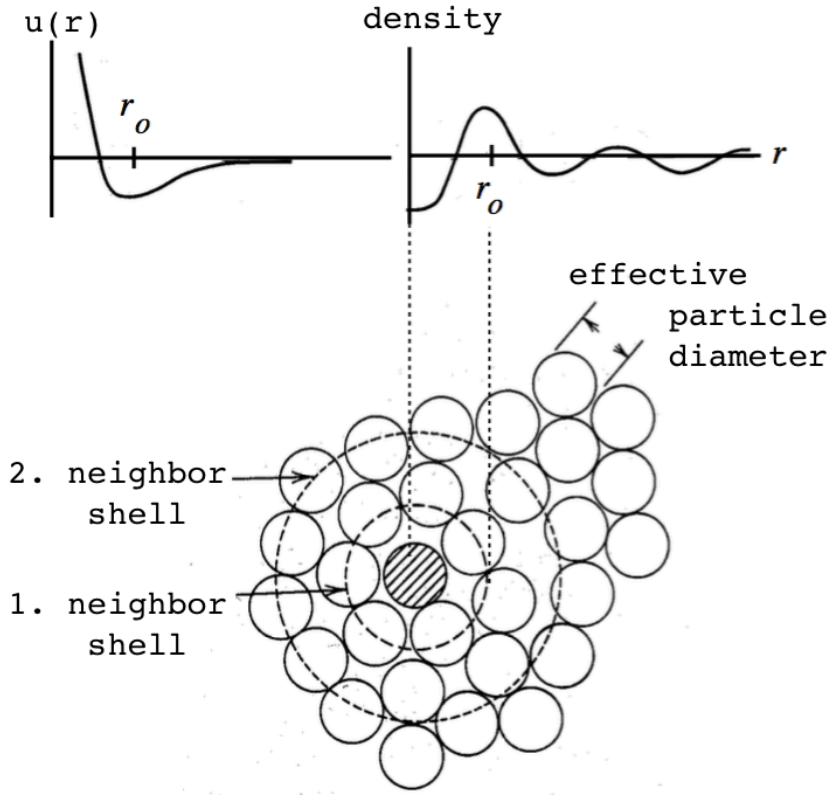


Figure 5: Relation between inter-particle potential and the ordering imposed by the shaded particle on its neighborhood. The validity of Eq. (10) is based on the assumption that the density variations beyond r_{cut} are negligible. Notice that $r_{cut}/\sigma = 3$ essentially means that this should be true after the third neighbor shell.

$$\vec{F}_i = \sum_{j(\neq i), r_{ij}^{min} \leq r_{cut}} \vec{f}_{ij} \ . \tag{12}$$

Comparing to the real world

All our simulations are carried out on the basis of Eq. (9). For example, in one of the projects you will use Monte Carlo simulations to find the gas-liquid critical point of a particle system. Let $T_{LJ,c}$ be your result for the critical temperature and $\rho_{LJ,c}$ your result for the critical density. A good place to look up corresponding numbers for real fluids like Argon or Methane is Ref.[4] Let's assume we look up the critical parameters for Methane, i.e. $T_{CH_4,c}$ and $\rho_{CH_4,c}$. How are these two numbers related to your simulation results? Notice that $k_B T_{CH_4,c}$ is a thermal energy. k_B of course is Boltzmann's constant. Because in our simulation energy is in units of ϵ, we should also measure this energy in ϵ. Thus

$$\frac{k_B T_{CH_4,c}}{\epsilon} = T_{LJ,c} \ . \tag{13}$$

With some background in Statistical Mechanics, we can see this even more clearly. Consider the Boltzmann factor $\exp[-U/(k_B T)]$. With $U/\epsilon = U_{LJ}$ we have

$$\exp\left[-\frac{U}{k_B T}\right] = \exp\left[-\frac{U_{LJ}}{k_B T/\epsilon}\right] = \exp\left[-\frac{U_{LJ}}{T_{LJ}}\right] \ . \tag{14}$$

The number density $\rho_{CH_4,c}$ has the unit [length^{-3}]. Because we measure length in units of σ, we find

$$\sigma^3 \rho_{CH_4,c} = \rho_{LJ,c} \ . \tag{15}$$

This means that if we know ϵ and σ for methane, i.e. ϵ_{CH_4} and σ_{CH_4}, then we can use Eq. (13 and (14) to compute $T_{CH_4,c}$ and $\rho_{CH_4,c}$ from our simulation results $T_{LJ,c}$ and $\rho_{LJ,c}$. If we do not yet know ϵ_{CH_4} and σ_{CH_4}, then we can use the same equations to compute them based on the experimental values for $T_{CH_4,c}$ and $\rho_{CH_4,c}$ from Ref.[4]

Ref.[4] also provides the critical pressure of Methane, $P_{CH_4,c}$. How does the equation relating this pressure to the critical pressure obtained in the simulation, $P_{LJ,c}$ look like? The game is always the same. The LJ-quantities in our simulation are dimensionless. Because pressure has the dimension [energy/volume], we have

$$\frac{\sigma^3}{\epsilon} P_{CH_4,c} = P_{LJ,c} \; . \tag{16}$$

In the next section we discuss Molecular Dynamics simulations. Basically, we solve Newton's equation of motion for every particle i in the system:

$$m \frac{d^2 \vec{r}_i}{dt^2} = \vec{F}_i \; . \tag{17}$$

Here m is the mass of the particles. We can write the same equation in a dimensionless form, which we then can implement on the computer:

$$\frac{m\sigma}{\tau^2} \frac{d^2 \vec{r}_{LJ,i}}{dt_{LJ}^2} = \frac{\epsilon}{\sigma} \vec{F}_{LJ,i} \; . \tag{18}$$

Setting

$$\tau = \sqrt{\frac{m\sigma^2}{\epsilon}} \tag{19}$$

we find the desired equation of motion

$$\frac{d^2\vec{r}_{LJ,i}}{dt_{LJ}^2} = \vec{F}_{LJ,i} \; . \tag{20}$$

The quantity τ, as defined by Eq. (19), is the LJ unit of time in our simulations.

In order to get a feeling for τ, we again consider the methane molecule. Its mass is $m_{CH_4} = 16$ amu. Based on the critical point data we find $\sigma_{CH_4} \approx 3.7$ Å and $\epsilon_{CH_4}/k_B \approx 141$ K. This yields $\tau_{CH_4} \approx 1.4 \cdot 10^{-12}$ s! During this time a methane molecule in a gas, whose temperature is $T = 300$ K, travels the average distance $\sqrt{k_B T/m_{CH_4}}\tau_{CH_4} \approx \sigma_{CH_4}$. Notice that the square root is the thermal velocity. During a collision of two methane molecules, we must calculate the forces between them in steps much smaller than τ_{CH_4}. This is because the potential energy, and thus the force of interaction, may be very different even when the inter molecular distance changes by only a fraction of σ (cf. Fig. 4). In practice a save timestep in a MD simulation of a LJ system is roughly 0.001τ. Thus, we not only have few particles in our simulation boxes, we can follow their dynamics only for a very brief time interval by macroscopic standards. Analogously, the corresponding maximum translation of a molecule during a MC step should be much less than σ.

Table 1: Conversion to and from LJ systems

U/ϵ	U_{LJ}
$k_B T/\epsilon$	T_{LJ}
$\sigma^3\rho$	ρ_{LJ}
$\sigma F/\epsilon$	F_{LJ}
t/τ	t_{LJ}

From now on we no longer use the index LJ to indicate (dimensionless) LJ quantities. Unless explicitly stated otherwise all quantities in the remainder of this tutorial are LJ quantities!

Molecular Dynamics simulations

Theoretical background

In a MD simulation we numerically solve Newton‘s equations of motion for a many-particle system. Particles may be atoms or molecules. This is remarkable, because we have learned that this scale is governed by quantum mechanics.

We are dealing with gases and liquids, for which classical mechanics works quite well under the condition

$$\rho^{-1/3} \gg \Lambda_T \tag{21}$$

(cf. section 5.2 in Ref.[5]). Here $\rho = N/V$ is the number density and $\Lambda_T = \sqrt{h^2/(2\pi m k_B T)}$, where h is Planck‘s constant, m is the particle mass, k_B is Boltzmann‘s constant, is the thermal wavelength. Notice that Eq. (21) is one of the exceptions from our above rule regarding LJ quantities. Notice also that $\Lambda_T \approx 17.5\text{Å}/\sqrt{mT}$, where we use atomic mass units and Kelvin for the temperature. For instance, for Methane we have $m_{CH_4} = 16$ amu and if $T = 300$ K and thus $\Lambda_T \approx 0.25$ Å. The average center of mass separation, i.e. $\rho^{-1/3}$, between methane molecules at the critical point is ≈ 5.5 Å. This means that in the case of methane the above condition is satisfied.

The result of a MD simulation is the so called trajectory, i.e. file containing the particle coordinates, q_i, and momenta, p_i, collected over $k = 1$ to K timesteps, Δt:

$$\{q_i(t_k), p_i(t_k)\}_{i=1,k=1}^{3N,\mathrm{K}} \, . \tag{22}$$

With this information we can estimate the (time) average of any quantity A, i.e.

$$\langle A \rangle = \lim_{t\to\infty} \frac{1}{t} \int_0^t dt' A(\{q(t), p(t)\}, t) \, , \tag{23}$$

which may depend on either the coordinates or the momenta or on both (cf. Eq. (3)).

How it‘s done

A simple integration algorithm for Eq. (20) can be constructed as follows. We start with the series expansions of the position vector of particle i, $\vec{r}_i(t)$, and its velocity, $\vec{v}_i(t)$:

$$\vec{r}_i(t+\Delta t) = \vec{r}_i(t) + \Delta t\vec{v}_i(t) + \frac{1}{2}\Delta t^2\vec{F}_i(t) + \mathcal{O}(\Delta t^3) \tag{24}$$

$$\vec{v}_i(t+\frac{\Delta t}{2}) = \vec{v}_i(t) + \frac{\Delta t}{2}\vec{F}_i(t) + \mathcal{O}(\Delta t^2) \tag{25}$$

$$\vec{v}_i(t+\Delta t) = \vec{v}_i(t+\frac{\Delta t}{2}) + \frac{\Delta t}{2}\underbrace{\vec{F}_i(t+\frac{\Delta t}{2})}_{=\vec{F}_i(t+\Delta t)+\mathcal{O}(\Delta t)} + \mathcal{O}(\Delta t^2)\ . \tag{26}$$

Adding the last two equations, we obtain the final algorithm:

$$\vec{r}_i(t+\Delta t) \approx \vec{r}_i(t) + \Delta t\vec{v}_i(t) + \frac{1}{2}\Delta t^2\vec{F}_i(t) \tag{27}$$

$$\vec{v}_i(t+\Delta t) \approx \vec{v}_i(t) + \frac{\Delta t}{2}\left(\vec{F}_i(t+\Delta t) + \vec{F}_i(t)\right)\ . \tag{28}$$

The first line advances the position, we shall call its implementation MOVER, whereas the second line advances the velocity, we shall call its implementation MOVEV. By repeating MOVER and MOVEV we are able to collect the system‘s trajectory. The entries in our trajectory list are separated by the timestep Δt. Large Δt allow to follow the system's dynamic over a longer time. However, a large Δt also means a large numerical error (cf. Eqs. (24) to (26)). Thus, the actual timestep is a compromise, allowing a long trajectory with 'acceptable' error. Usually it is a good idea to vary the timestep and compare the attendant simulation results. A reasonable timestep for LJ-simulations is $\Delta t = 0.001$ (cf.above).

Notice that we need one force evaluation per timestep Δt only. The x-component

of the force is given by

$$F_{i,x} = -\sum_{j(\neq i)=1}^{N} \frac{d}{dx_i} u_{ij} = \sum_{j(\neq i)=1}^{N} f_{ij,x} \ . \tag{29}$$

A straightforward derivative yields

$$f_{ij,x} = 24 \left[2 r_{ij}^{min\,-13} - r_{ij}^{min\,-7} \right] \frac{x_{ij}^{min}}{r_{ij}^{min}} \ . \tag{30}$$

Notice that this is the x-component of the force on particle i exerted by the real particle j or its nearest image (The force curve in Fig. 4 is $24\left[2r^{-13} - r^{-7}\right]$). If for instance particle i is located at the origin and j (real or image) is far away along the positive x-axis, then $f_{ij,x}$ is positive, i.e. particle i is pulled towards j. In order to work out the other two components of the force we merely replace x by y and z.

NVE MD code

The core of an MD program looks like this:

$\boxed{Main}$

... generate initial configuration ...

FORCE (k=1)

Do k=2, NSTEP

    MOVER(k-1)

    FORCE(k)

    MOVEV(k-1)

End k

... output ...

The index k is the timestep index. The program integrates the equations of motion for NSTEP timesteps. The three subroutines FORCE, MOVER, and MOVEV do this:

$\boxed{FORCE(k)}$

Do i=1, N-1

Do j=i+1, N

If $r_{ij}^{min} < r_{cut}$ Then

$F_i = +f_{ij}$

$F_j = + - f_{ij}$

$(U = +u_{ij})$

End j

End i

$\boxed{MOVER(k)}$

Do i=1, N

$r_i(k+1) = r_i(k) + \Delta t v_i(k) + \frac{1}{2}\Delta t^2 F_i(k)$

End i

$\boxed{MOVEV(k)}$

Do i=1, N

$v_i(k+1) = r_i(k) + \frac{1}{2}\Delta t\big(F_i(k+1) + F_i(k)\big)$

End i

Notice that $= +$ means that the quantity on the left is incremented by the quantity on the right. Notice also that the above is a shorthand notation omitting vectors. MD

does not use the potential directly, this is why the calculation of the total potential energy U is optional. An example program set up according to this schema is included in the Appendix.

• **Exercise - FORCE for $N = 4$:** By going through the FORCE routine step by step (pretending $N = 4$ and $r_{cut} = \infty$) write down the force on each particle for every distinct pair i, j as a sum over pair-forces f_{ij}.

The line ... generate initial configuration ... requires some discussion. Initially the particles should be placed on the nodes of a (cubic) lattice and according to the required density. This is superior to random placement, because the latter may create overlapping particles or close contacts. After the random assignment of the initial velocities, it may be necessary to reset the center of mass velocity,

$$\vec{v}_{CM} = \frac{1}{N} \sum_{i=1}^{N} \vec{v}_i \ , \tag{31}$$

to zero. This is done in the single loop

Do i=1, N

$v_i = v_i - v_{CM}$

End i

Otherwise the unphysical translation of the center of mass can cause problems, e.g. a wrong temperature.

Adjusting temperature

The particle system modelled here has been supplied with a certain amount of energy in the program part called ... generate initial configuration .... The particles have random initial velocities and due to their initial positions also an attendant potential

energy. The total energy, i.e. $E = E_{kinetic} + U$, remains constant (sidestepping the issue of numerical errors). We therefore model what is called an NVE system, i.e. the particle number, N, the volume, V, and the total energy, E, are constant.

The system will undergo a transient period called *equilibration* already mentioned. Afterwards the instantaneous temperature,

$$T(k) = \frac{1}{3N} \sum_{i=1}^{N} v_i(k)^2 \, , \tag{32}$$

oscillates around a constant average value $\bar{T}$, as shown in the last figure in the appendix. This again is an estimate of the temperature in the system (cf. (5)). Notice that the above equation is based on $E_{kinetic}(k) = \frac{3}{2} N T(k)$, which follows from the *generalized equipartition theorem* (e.g., Ref.[2]).

Thus far the average temperature in the simulation is difficult to adjust to a particular value. A simple method for temperature adjustment is the heat-flux approach. Assume that the heat flux, J_Q, leads to the following change of the instantaneous kinetic energy between timesteps k and $k+1$:

$$J_Q = \frac{\Delta Q}{\Delta t} = \frac{1}{\Delta t} \sum_{i=1}^{N} \frac{1}{2} v_i^2(k)(\lambda^2 - 1) = \frac{1}{\Delta t} \frac{3}{2} N T(k)(\lambda^2 - 1) \, . \tag{33}$$

This ΔQ corresponds to the velocity rescaling

$$v_i(k) \to \lambda v_i(k) \tag{34}$$

after every timestep and for every particle. λ is slightly larger or smaller than one, depending on whether heat is flowing into or leaving the system.

If J_Q depends linearly on the difference between the instantaneous temperature, $T(k)$, and the target temperature, T_B, i.e.

$$J_Q = \alpha_T (T_B - T(k)) \ , \tag{35}$$

where α_T is a constant, then the combination of (33) and (35) yields

$$\lambda = \sqrt{1 + \frac{2\Delta t}{\tau_T}\left(\frac{T_B}{T(k)} - 1\right)} \approx 1 + \frac{\Delta t}{\tau_T}\left(\frac{T_B}{T(k)} - 1\right) \ . \tag{36}$$

Here $\tau_T^{-1} = \alpha_T/(3N)$ is another constant. $\Delta t/\tau_T$ should be small, so that the continuous rescaling of the velocities (34) leads to a gradual approach of $T(k)$ towards T_B (More details can be found in section III of Ref.[3] In particular it is shown that τ_T is a typical relaxation time needed to adjust the current temperature to the value T_B).

• **Exercise - temperature adjustment in MD:** Recreate the NVE MD program in the appendix. Introduce the above temperature rescaling and show a plot of $T(t)$ vs. time t, which illustrates the approach of $T_B = 4$ from a different equilibrium temperature $T_o \approx 2$. Fit the function

$$T(t) = T_B - (T_B - T_o)\exp[-2t/\tau_{T,fit}] \ , \tag{37}$$

to your data, where $\tau_{T,fit}$ is an adjustable parameter. Compare $\tau_{T,fit}$ to τ_T used in your simulation.

A practical example: the coefficient of thermal conductivity

The diffusive heat flux through a wall of area A and thickness d is given by

$$\frac{dQ}{dt} = \lambda_{TC} \frac{A}{d} \Delta T \; . \tag{38}$$

The quantity ΔT is the temperature difference on the two sides of the wall and λ_{TC} is the coefficients of thermal conductivity. When the wall material is 'dry air' (insulation material is mainly dry air), then $\lambda_{TC} \approx 0.03$ W/(m K) in MKSA units. If the wall is copper, however, then $\lambda_{TC} \approx 400$ W/(m K).

In a particle simulation you can calculate λ_{TC} via

$$\lambda_{TC} = \frac{V}{3T^2} \int_0^\infty dt \langle \vec{J}(0) \cdot \vec{J}(t) \rangle \; , \tag{39}$$

where V is the volume of the simulation box. The quantity $\langle \vec{J}(0) \cdot \vec{J}(t) \rangle$ is an auto-correlation function (cf. Eq. (6)). The definition of the diffusive heat current, $\vec{J}(t)$, is

$$\vec{J}(t) = \frac{1}{V} \sum_{i=1}^{N} \left[\vec{v}_i(t) \delta e_i(t) + \frac{1}{2} \sum_{j(\neq i)=1}^{N} \vec{r}_{ij} (\vec{f}_{ij} \cdot \vec{v}_i) \right] \; , \tag{40}$$

where $\delta e_i(t) = e_i(t) - \bar{e}(t)$ is the total energy (kinetic plus potential) of particle i, i.e. $e_i(t) = \frac{1}{2} v_i^2(t) + U_i(t)$, minus the average energy per particle in the system, i.e. $\bar{e}(t)$. A derivation of Eq. (39) can be found in Ref.[6] Two examples showing $\langle \vec{J}(0) \cdot \vec{J}(t) \rangle$ are depicted in Fig. 6 (you will calculate these auto-correlation functions yourself in the project outlined below). Notice that the auto-correlation function is calculated via

$$\langle \vec{J}(0) \cdot \vec{J}(t) \rangle \approx \frac{1}{K-k} \sum_{i=1}^{K-k} \vec{J}_i \cdot \vec{J}_{i+k} \; , \tag{41}$$

where $t = \Delta t \, k$. For large k, i.e. when k is not too different from K, the number of

terms in the average becomes small, which enhances the scatter. The integral in Eq. (39) may be approximated by the sum over all values of the auto-correlation function multiplied by Δt. Fig. 7 shows the result based on the left auto-correlation function in the previous figure. Notice that $\Delta t\, k_{max}$ is the upper limit of the integral in Eq. (39). The auto-correlation function has decayed after about one LJ time unit corresponding to $k_{max} = 1000$. If k_{max} is increased beyond this value $\lambda_{TC}(k_{max})$ remains constant within fluctuations. In the case of the right auto-correlation function depicted in Fig. 6 it is harder to determine λ_{TC}. This is because the auto-correlation function decays much more slowly. What used to be a narrow peak at about $t = 0.1$ now is a plateau-like shoulder persisting out to large t. In fact, there are thermodynamic conditions for which the decay of the auto-correlation function is so slow (long-time-tail), that with the methods at our present disposal we cannot determine λ_{TC} at all!

Question: Fig. 7 looks very much like the 'mock series of data points' in Fig. 2. Can we apply the method discussed above to determine the standard error of λ_{TC}? What are the two main conceptual differences between the figures?

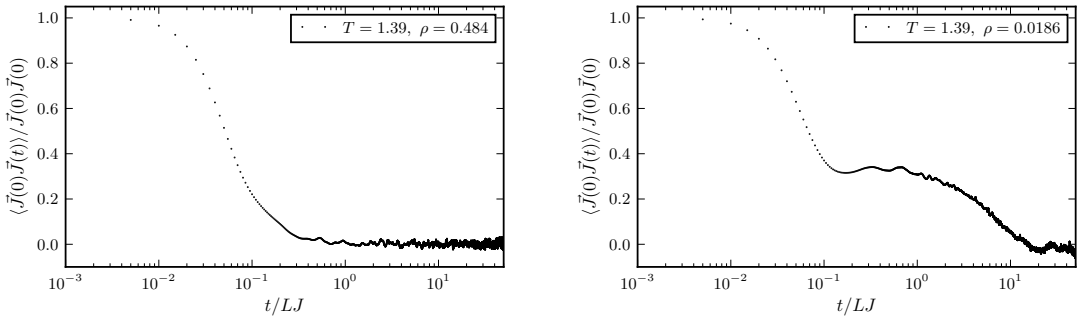


Figure 6: The two JJ auto-correlation functions corresponding to the thermodynamic conditions specified in table 2.

.

• **Project - thermal conductivity:** Table 2 lists two selected values for λ_{TC} at different conditions for Argon. You should try to obtain corresponding theoretical values from your MD program. You need to look up the critical point data for

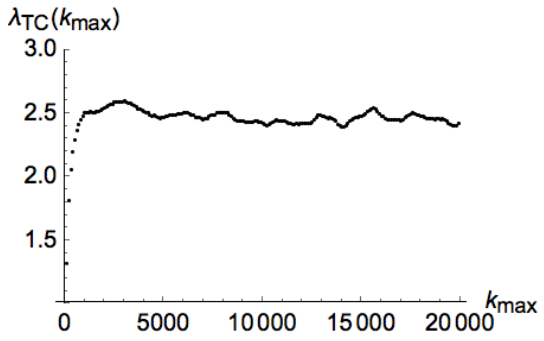


Figure 7: λ_{TC} based on the left auto-correlation function in the previous figure. Notice that $\Delta t\, k_{max}$ is the upper limit of the integral in Eq. (39).

.

Argon to convert the units as discussed above (e.g. Ref.[4]). Assume that in the LJ system $\rho_c = 0.31$ and $T_c = 1.32$ (You are supposed to check this in another exercise below. For the moment you may use these values.). Make sure that you understand how to convert the data in the table to LJ units - do this before you proceed!

Hints:

(a) You can use the MD program in the appendix together with the extension that adjusts the temperature. After the temperature has reached the desired value, you should switch to a large $\tau_T (\approx 1000)$ in order to minimize perturbation of your system. In essence you want to be as close to NVE as possible.

(b) In principle you can store all quantities needed to calculate $\vec{J}(t)$ according to Eq. (40) in a file. When the MD has finished, you analyse this file. In praxis, however, this is not feasible, because the double sum $\vec{J}_F \equiv \sum_{i=1}^{N} \sum_{j(\neq i)=1}^{N} \vec{r}_{ij}(\vec{f}_{ij} \cdot \vec{v}_i)$ requires all pairwise forces to be stored at every timestep. Thus you better do this calculation 'on the fly', i.e. while you do the actual MD simulation. The following subroutine in addition to FORCE is useful:

$\boxed{JF(k)}$

Do i=1, N-1

Do j=i+1, N

$fvi = f_{ij,x}v_{i,x} + f_{ij,y}v_{i,y} + f_{ij,z}v_{i,z}$

$fvj = f_{ij,x}v_{j,x} + f_{ij,y}v_{j,y} + f_{ij,z}v_{j,z}$

$JF_x = +x_{ij}^{min}\,(fvi + fvj)$

$JF_y = +y_{ij}^{min}\,(fvi + fvj)$

$JF_z = +z_{ij}^{min}\,(fvi + fvj)$

End j

End i

Notice that all quantities used here are computed at timestep k, i.e. this subroutine should be inserted following MOVEV(k-1) in the code on page 17. Before advancing the timestep, you are now ready to write the coordinates, $x_i(k)$, $y_i(k)$, $z_i(k)$, the attendant velocity components, the potential energy of particle i, $U_i(k)$, as well as the components $JF_x(k)$, $JF_y(k)$, $JF_z(k)$ to a file for later analysis - which means the calculation of the auto-correlation function as well as λ_{TC}. The necessary calculation of $U_i(k)$ can be integrated into FORCE(k) analogous to the calculations of F_i and F_j (notice: $u_{ij} = u_{ji}$ whereas $f_{ij} = -f_{ji}$).

(c) The sample average of the energy per particle, $\bar{e}$, should look like this $\bar{e} = \frac{1}{KN}\sum_k \sum_i e_i(k)$, i.e. include all particles at every timestep.

Table 2: Experimental values of λ_{TC} at two different conditions for Argon from Ref.[4] (6-22 to 6-23). Notice that the pressures provided here are not needed in the simulation.

T [K]	P [MPa]	ρ [mol/L]	$\lambda_{TC}^{(exp)}$ [10^{-3} W/(mK)]
160	1.0	0.799	11.1
160	10	20.816	52.8

Monte Carlo Simulation

Theoretical backround

The Molecular Dynamics technique produces system trajectories (22), whereas the Monte Carlo technique produces system configurations

$$\{q_i(k)\}_{i=1,k=1}^{3N,\mathrm{K}} \,. \tag{42}$$

Time does no longer appear. The index k refers to the kth step of the MC algorithm. Specifically, there is no connection between $q_i(t_k)$ and $q_i(k)$. Likewise, K is generally different in MD and MC.

The list of configurations can be used to calculate sample averages of any quantity which solely depends on the generalized coordinates q_i. An example is the total potential energy U:

$$\bar{U} = \frac{1}{K}\sum_{k=1}^{K} U(k) \,. \tag{43}$$

The calculation of statistical errors and correlation functions is analogous to Molecular Dynamics. Here we merely replace 'timestep' by 'MC step' and 'trajectory' by 'configuration list'. The difference is that the decay of correlations is physically meaningful in MD. In fact, we use them to calculate transport coefficient, which we cannot do on the basis of MC simulations.

Depending on how we set up the MC algorithm we can calculate averages in different ensembles. For instance, if the number of particles, N, the volume of the simulated system, V, and the temperature in the system, T, are held constant, then we call this a NVT-ensemble. Depending on the problem at hand, other ensembles, i.e. other thermodynamic quantities are held constant, may be better for a specific purpose.

The central part of every so called Metropolis MC algorithm consists of the following two items:

1. Generate a new random system configuration. This should be based on a random change, called *move*, of the previous configuration in the list. A small random displacement of a randomly selected particle is an example.
2. Evaluate the Metropolis criterion. If the criterion is *true* then add the new configuration to the list (42). If the criterion is *false* then add the previous configuration, already in the list, once again to the list. Then goto 1 and continue until $k = K$.

The Metropolis criterion is simply

$$\min\left(1, \frac{p_{new}}{p_{old}}\right) \geq \mathrm{RND}(0,1)\ . \tag{44}$$

$\min(a, b)$ returns the smaller of the two numbers a and b and $\mathrm{RND}(0, 1)$ is a random number on (0,1). The quantity p is the statistical probability of the system configuration; p_{new} refers two the new configuration and p_{old} refers to the last configuration stored in the list (42).

There are two questions at this point. (a) How does one find the right p for the problem at hand? (b) How does one know whether or not the above two-step algorithm really works?

Let us focus on question (b) first. In order to show that the algorithm indeed works, it is best to reduce the number of possible system configurations. Hundreds of particles in a simulation box may be arranged in almost infinitely many configurations. In this case it is hard to follow what the algorithm really does. It is best to construct an example with say only four possible configurations called 1, 2, 3 and 4.

The following is a list of these configurations analogous to (42):

$$12123412123341443232\ . \tag{45}$$

Step 1 in our MC algorithm would be

1. Randomly select one of the three numbers $-1, 0, +1$ and add this number to the last *configuration* in the above list. In order to limit our configuration space to 1, 2, 3, and 4 do the following. If the new number (or configuration) is 0 then use 4 instead. If the new number (or configuration) is 5 then use 1 instead. This is quite analogous to particle simulations with periodic boundaries.

Repeating this infinitely many times yields a configuration list in which 1, 2, 3, and 4 appear equally often, i.e. $p(1) = p(2) = p(3) = p(4)$. In this case $p(1) = p(2) = p(3) = p(4) = 1/4$. The Metropolis criterion does not do anything and can be omitted.

However, let us make up a different set of probabilities. For instance, $p(1) = p(3)$ and $p(2) = p(4)$ but $p(2) = 2p(1)$. This means that we want the even numbers to appear twice as often on average compared to the odd numbers. The example is so easy that we can work out the probabilities right away: $p(1) = 1/6$, $p(2) = 2/6$, $p(3) = 1/6$, and $p(4) = 2/6$. The following program code in *Mathematica* is an implementation of Metropolis MC for this case:

```
"Test of Metropolis MC";

"this list counts the occurrences of the four numbers"; h = {0, 0, 0, 0};
```

```
"target probability distribution"; p = {1/6, 2/6, 1/6, 2/6};
"max. number of MC steps"; K = 10000;
"initial configuration"; j = RandomInteger[{1, 2}];
jold = j;
Do[j+=RandomInteger[{-1, 1}]; If[j == 0, j = 4]; If[j == 5, j = 1];
If[Min[1, p[[j]]/p[[jold]]] ≥ RandomReal[], {h[[j]]+=1, jold = j}, {h[[jold]]+=1, j = jold}],
{k, 1, K}];
Print[N[h/K]]
```

```
{0.1641, 0.3406, 0.1655, 0.3298}
```

The list at the end shows the relative frequencies of 1, 2, 3, and 4 in a series of length 10000 generated by Metropolis MC. The exact result would have been {0.16666.., 0.33333.., 0.16666.., 0.33333..}. We can improve the agreement by simply generating a (much) longer configuration list. It is important to note that the Metropolis criterion uses probability ratios, because it is much harder and mostly impossible to calculate the full normalized probabilities in particle simulations.

A much more detailed discussion of this example and a description of how Metropolis works in this case can be found in Ref.[1] (chapter 6) or Ref.[2] (chapter 7).

• **Exercise - biased sampling and faulty Metropolis criterion:** Repeat the above *Test of Metropolis MC* with the following modification:

(a) Instead of *...if the new number (or configuration) is 0 then use 4...* use *...if the new number (or configuration) is 0 then use 2....* This replaces one of the periodic boundaries by a reflective boundary. What is your result after 10^6 MC steps?

(b) Instead of...

...if the Metropolis criterion is *true* then add the new configuration to the configuration list. If the criterion is *false* then add the previous configuration, already in the list, once again to the list...

...use...

...if the Metropolis criterion is *true* then add the new configuration to the configuration list. If the criterion is *false* then add nothing to the configuration list...

... What is your result after 10^6 MC steps?

• **Exercise - random number generator test:** Monte Carlo simulations require vast quantities of independent random numbers. In order to avoid simulation errors due to correlated random numbers you must test your random number generator. A simple but nonetheless good test is the following. Generate a sequence of 2×10^7 random numbers $\xi \in (-1, 1)$. Produce a x-y-point plot of subsequent pairs, i.e. $\ldots$, $(x = \xi_i, y = \xi_{i+1})$, $(x = \xi_{i+2}, y = \xi_{i+3})$, $\ldots$, extracted from this series. The result should by an almost uniformly shaded square without a discernible systematic pattern.

Now we return to the first question, i.e. question (a). How does one find the proper p for simulation boxes containing Lennard-Jones particle systems? We start with the famous formula

$$S = \ln \Omega \,, \tag{46}$$

relating the entropy, S, to the number of microstates, Ω, in the universe. A microstate describes a distinct and spontaneously possible 'arrangement' of all things.

If I decide to fix the value of some quantity X inside a small corner of the universe, a system, to the value x, then this imposes a constraint. The latter reduces Ω and thus S. By constraining X to some value, I rule out certain arrangements of things, i.e. microstates, violating the constraint. The attendant entropy reduction is

$$\Delta S = \ln \Omega' - \ln \Omega \,. \tag{47}$$

Here Ω' is the number of possible microstates in the universe under the condition that X in the system is x. In other words, the probability that the universe spontaneously and without my interference decides to realise x inside the system is

$$p(x) = \frac{\Omega'}{\Omega} = \exp[\Delta S] \tag{48}$$

Specifically Ω' is the product of the number of realizable microstates inside the system and the number of realizable microstates outside the system,

$$\Omega' = \Omega_{out}(x_{univ} - x)\Omega_{system}(x) \ . \tag{49}$$

Inside the system X has the value x, outside the system X is $x_{univ} - x$. For instance, X could be the internal energy, E, in a closed system or the number of particles, N, if the system is open. Because $x_{univ} \ll x$ we can expand $\Omega_{out}(x_{univ} - x)$ or rather $\ln \Omega_{out}(x_{univ} - x)$ at $X = x_{univ}$, i.e.

$$\ln \Omega_{out}(x_{univ} - x) \approx \ln \Omega_{out}(x_{univ}) - \frac{\partial \ln \Omega_{out}(X)}{\partial X}\Big|_{X=x_{univ}} x \ . \tag{50}$$

Thus we find

$$p(x) \propto \Omega_{syst}(x) \exp\left[-\frac{\partial S_{out}(X)}{\partial X}\Big|_{X=x_{univ}} x\right] \ . \tag{51}$$

If the system is in equilibrium with its surrounding (not necessarily as big as the entire universe), then we know what the partial derivative is. From the combination of the first and second law of thermodynamics (see for instance Ref[1] section 1.4.3) follow

$$\left.\frac{\partial S}{\partial E}\right|_{V,N,..} = \frac{1}{T} \tag{52}$$

$$\left.\frac{\partial S}{\partial V}\right|_{E,N,..} = \frac{P}{T} \tag{53}$$

$$\left.\frac{\partial S}{\partial N}\right|_{E,V,..} = -\frac{\mu}{T} \tag{54}$$

$$\vdots$$

Notice that the subscript *out* is not necessary, because the above is true for the entropy in any equilibrium system. Thus, if X is the internal energy of our system, i.e. $x = E$, we have

$$p(E) \propto \Omega_{syst}(E) \exp\left[-(1/T)E\right] \ . \tag{55}$$

If on the other hand X is the number of particles in our system, i.e. $x = N$, we have

$$p(N) \propto \Omega_{syst}(N) \exp\left[(1/T)\mu N\right] \ , \tag{56}$$

where μ is the chemical potential. More generally, if we consider a system in which both the internal energy and the particle number are variable then

$$p(E,N) \propto \Omega_{syst}(E,N) \exp\left[-(1/T)(E-\mu N)\right] \ . \tag{57}$$

Because the Metropolis criterion works with probability ratios we need not worry about the proportionality constants. Following this reasoning we can construct p for every relevant situation.

There is only the factor Ω_{syst}, which we need to worry about. In classical mechanics

(cf. (22)) we can measure the position and momentum of a particle, defining its microstate, at any time with arbitrary precision. But Heisenberg's uncertainty principle tells us, that this is not really true. This means that phase space is discretised in cells of size $(\Delta x \Delta p_x)/h \sim 1$, where h is Planck's constant. These cells now define the possible microstates. If our volume is $V = L^3$, where L is the linear dimension, then we see that doubling the linear dimension, i.e. $L \to 2L$, also doubles the number of cells along this direction and thus $\Omega_{syst} \propto V^N$. In addition, we must worry about the distinguishability of microstates. Exchange of two particles looking exactly alike does produce nothing new. This can be included by an extra factor $N!^{-1}$. All in all this implies

$$\Omega_{syst} \propto \frac{V^N}{N!} \ . \tag{58}$$

In the following we consider an example where this form of Ω_{syst} is sufficient, because everything else cancels when we compute the probability ratios in the Metropolis criterion.

Remark: It is assumed that the MC moves do not affect the kinetic energy. This means that $\exp[-E/T]$ is replaced by $\Lambda_T^{-3N} \exp[-U/T]$. The factor Λ_T^{-3N} results from the integration over the momenta in the system. In the following example it cancels out. But in other cases, e.g. the MC simulation of adsorption, we must include it.

A practical example: Gas-liquid phase coexistence

Fig. 8 depicts the phase diagram of a simple liquid [1]. Our goal is the calculation of the phase coexistence line between gas and liquid including the determination of the critical point. For this purpose we set up two simulation boxes initially containing about 100 particles each. The particles are located on cubic lattices. The lattice constants are chosen so that the initial number density, $\rho = N/V$, in both boxes is 0.3.

[1] If you are not familiar with simple phase diagrams, in particular gas-liquid phase coexistence, you should read sections 4.1 and 4.2 in Ref. [1]

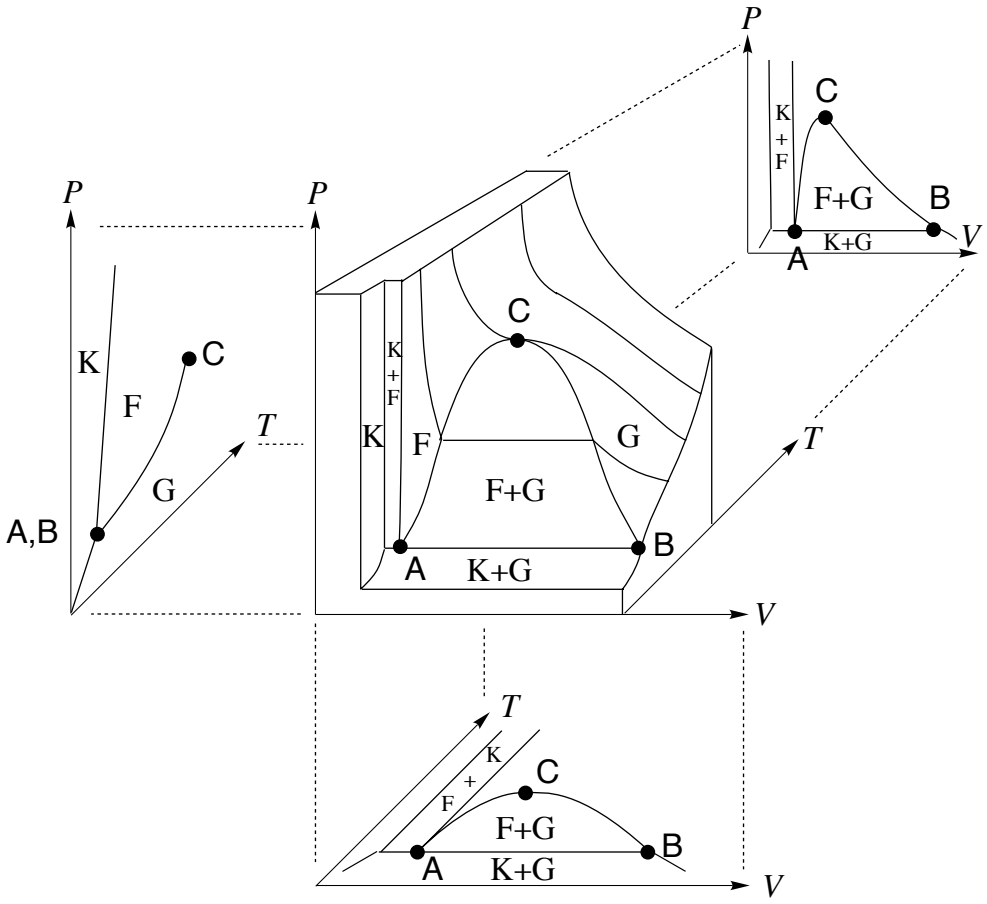


Figure 8: Phase diagram of a simple liquid and its projections on different planes. Here the letters G, F, and K stand for gas, liquid and solid, respectively. The notation ...+... indicates coexistence regions between the respective phases. C indicates the gas-liquid critical point.

By variation of the respective particle densities, using a suitable MC algorithm, we expect one box to approach the gas density at coexistence, ρ_g, whereas the other box approaches the liquid density at coexistence, ρ_l. Both densities are functions of temperature. By variation of the temperature we can trace out the entire coexistence curve.

The thermodynamic states of the two simulation boxes are characterized by six variables: T_g, T_l, P_g, P_l, μ_g, and μ_l, i.e. temperature, pressure, and chemical potential in each box. However, phase coexistence is an equilibrium phenomenon which imposes constraints. There is thermal equilibrium, which means $T_g = T_l (= T)$. There is meachnical equilibrium, which means $P_g = P_l$. And finally there is chemical equlibrium, i.e. $\mu_g = \mu_l$. These three constraints reduce the number of independent quantities to 3. But in reality we can vary only one quantity on the coexistence line. If we vary T, then P, ρ_g, and ρ_l are fixed. Thus, we need two additional constraints. Here we choose $V = V_1 + V_2 = constant$ and $N = N_1 + N_2 = constant$, where the indices refer to the boxes. This means in particular, when we change the box volumes and particle numbers in order to approach the densities at coexistence, $\Delta V_1 = -\Delta V_2$ and $\Delta N_1 = -\Delta N_2$.

The probability needed in the Metropolis criterion is given by

$$p = \prod_{\nu=1}^{2} p_\nu(E_\nu, V_\nu, N_\nu) \propto \prod_{\nu=1}^{2} \frac{{V_\nu}^{N_\nu}}{N_\nu!} \exp\left[-(1/T)(E_\nu + PV_\nu - \mu N_\nu)\right] \ . \tag{59}$$

Because of the above constraints on the total volume and the total particle number this simplifies to

$$p \propto \prod_{\nu=1}^{2} \frac{{V_\nu}^{N_\nu}}{N_\nu!} \exp\left[-(1/T)U_\nu\right] \ , \tag{60}$$

where we also use that all moves affect the potential energy only. Now we are ready to set up the MC algorithm.

Gibbs-Ensemble Monte Carlo (GEMC) algorithm

1. **translation move in box 1**: select particle i at random and calculate new random position via $\vec{r}^{(1)}_{i,new} = \vec{r}^{(1)}_{i,old} + \delta r(\xi_x, \xi_y, \xi_z)$. Here δr is a (small) constant maximum displacement and the ξ are independent random numbers on (-1,1).

2. **Metropolis - translation move in box 1**: probability ratio

$$\frac{p_{new}}{p_{old}} = \exp\left[-(1/T)\Delta U\right] \ ,$$

   where $\Delta U = U_{1,new} - U_{1,old}$.

3. **translation move in box 2**: select particle j at random and calculate new random position via $\vec{r}^{(2)}_{i,new} = \vec{r}^{(2)}_{i,old} + \delta r(\xi_x, \xi_y, \xi_z)$. Notice that every random numbers ξ is always independent from every other ξ!

4. **Metropolis - translation move in box 2**: probability ratio for the Metropolis criterion as described above is

$$\frac{p_{new}}{p_{old}} = \exp\left[-(1/T)\Delta U\right] \ ,$$

   where $\Delta U = U_{2,new} - U_{2,old}$.

5. **volume move**: $\Delta V = \delta V \xi$, $V_{1,new} = V_{1,old} + \Delta V$ and $V_{2,new} = V_{2,old} - \Delta V$.

   Rescaling of particle positions via

   $\vec{r}^{(1)}_{i,new} = (V_{1,new}/V_{1,old})^{1/3} \ \vec{r}^{(1)}_{i,old} \ \forall i \in \{1, N_1\}$

and

$\vec{r}^{(2)}_{j,new}$=$(V_{2,new}/V_{2,old})^{1/3}$ $\vec{r}^{(2)}_{j,old}$ $\forall j \in \{1, N_2\}$.

A useful maximum volume change is $\delta V = 10^{-3}V$.

6. **Metropolis - volume move**: probability ratio

$$\frac{p_{new}}{p_{old}} = \left(1 + \frac{\Delta V}{V_{1,old}}\right)^{N_1} \left(1 - \frac{\Delta V}{V_{2,old}}\right)^{N_2} \exp\left[-(1/T)\Delta U\right] \ ,$$

where $\Delta U = U_{1,new} - U_{1,old} + U_{2,new} - U_{2,old}$.

7. **transfer move from box 1 to box 2**: select particle i in box 1 at random and calculate new random position in box 2, $\vec{r}^{(2)}_{i,new}$.

8. **Metropolis - transfer from 1 to 2**: probability ratio

$$\frac{p_{new}}{p_{old}} = \frac{V_2 N_{1,old}}{V_1 N_{2,old} + 1} \exp\left[-(1/T)\Delta U\right] \ ,$$

where $\Delta U = U_{1,new} - U_{1,old} + U_{2,new} - U_{2,old}$.

9. **transfer move from box 2 to box 1**: select particle j in box 2 at random and calculate new random position in box 1, $\vec{r}^{(1)}_{j,new}$.

10. **Metropolis - transfer from 2 to 1**: probability ratio

$$\frac{p_{new}}{p_{old}} = \frac{V_1 N_{2,old}}{V_2 N_{1,old} + 1} \exp\left[-(1/T)\Delta U\right] \ ,$$

where $\Delta U = U_{1,new} - U_{1,old} + U_{2,new} - U_{2,old}$.

11. **Continue** return to 1. if current number of MC steps less than K.

Fig. 9 depicts a snapshot taken during a GEMC simulation. At the start the red particles where in one box and the blue ones in the other. However, irrespective of their color they are all the same LJ particles.

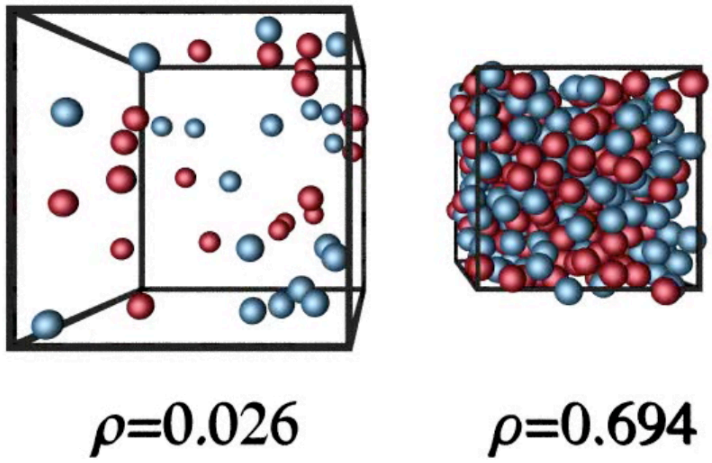


Figure 9: Snapshot of the simulation boxes in a GEMC simulation. Left: gas; right: liquid box. The red and blue particles initially were in different boxes.

• **Exercise - comparison between MD and MC:** Run the above GEMC code using a density of $\rho = 0.3$ in each box. Allow translation moves, including their attendant Metropolis criteria, only. Determine the equilibrium potential energies in the two independent boxes for $T = 2$. Using your MD program also compute the equilibrium potential energy for $\rho = 0.3$ and $T = 2$. Compare all three potential energies.

• **Project - gas-liquid coexistence:** Using a programming language of your choice write a Gibbs-Ensemble MC computer code and obtain the equilibrium densities $\rho_g(T)$ and $\rho_l(T)$, including error bars, for $1.1 \leq T < T_c$ (cf. Fig. 6.2 in Ref.[1]). The interaction potential is the Lennard-Jones potential. Determine both the critical temperature, T_c, and is the critical density, ρ_c, with the method described in Ref.[3] (starting on page 85). Look up gas-liquid coexistence densities in the literature and plot them in the T-ρ-plane together with your simulation results and the theoretical fit.

Concluding remarks

Which simulation method is better - Molecular Dynamics or Monte Carlo? When I started learning how to do molecular computer simulations my world was divided in two factions. There were those advocating the use of MD and others who favored MC. These discussions were as sensible as the discussion of the question whether a hammer is a better tool than a screwdriver. The answer depends on the problem - and this is true also in the case of MD vs. MC. Even though there probably is more overlap between the respective ranges of applications. The first programming project, thermal conductivity, is a problem in the domain of MD. Transport phenomena, where time is of central importance, do belong to this domain. Phase equilibria are generally easier to simulate using MC. MC has the distinct advantage that its moves may be unphysical, which can make them very efficient. This is because the results are equilibrium state functions and it does not matter what path we use to obtain them. One example for an unphysical move is the particle transfer between boxes. But there is also a risk here. If the density is high, then particle transfer becomes inefficient. In general most basic MC algorithms are easy to program - only the potential energy is needed. On the other hand, they do require probably more theoretical background in Statistical Mechanics. But then again - it is not too difficult to switch between ensembles. This is not true in the case of MD. On the other hand, when systems get complex, e.g. macromolecular systems, or if their density is high, the design of efficient MC moves can be difficult. In addition, there is always the danger of unwittingly introducing a bias leading to incorrect sampling.

I could continue listing pros and cons but in the end it comes down to experience and practice!

One last comment on computational effort. The FORCE routine contains a double-loop and at least the calculation of the minimum image distance is an operation $\mathcal{O}(N^2)$. Here we work with small systems - not larger than 100 to 300 particles. Thus we do not need special methods improving efficiency (cf. Ref.[3]). MC-particle moves involve

one particle only (except in the case of volume changes). You should therefore avoid computation of the potential energy of the entire system at every step. Instead keep track of the potential energy and only update the contribution due to the moved particle. As mentioned above, you can choose whichever programming language suits you best. You must be aware, however, that algebraic languages, like *Mathematica* used in the appendix, usually are vastly inferior, in terms of speed, to numerical languages when it comes to molecular simulations!

Remarks on how to present your work

Present your work in the form of a 30 min talk consisting of not more than 20 slides. One slide per exercise should be sufficient, with perhaps one exception where you need two, to state the problem and give the answer. Assume that your audience also has read this tutorial, i.e. do not include a general introduction to computer simulation. Focus on the exercises and the presentation of the two projects. Here you should include the necessary background information. Do not show computer code. Avoid fillers like 'computer simulation techniques are useful tools'. Every graph, formula, and statement you present must have a purpose and/or convey a message. Make sure that the content of every slide is clearly legible (size as well as color) - even from the back of the room. It is not a bad idea to give a practice talk to a (critical) friend using the original equipment. Aside from the slides for this talk no additional report is necessary.

Acknowledgment

I like to thank Sven Engelmann for his critical reading of this tutorial and, particularly, for pointing out a programming mistake in calculation of the heat current. He also supplied me with Fig. 6.

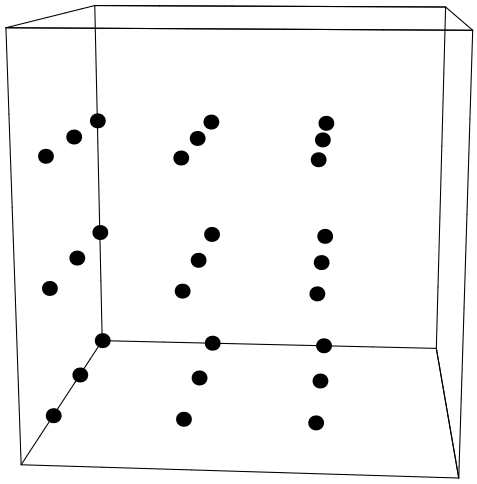
Appendix: *Mathematica* program for MD in the NVE-ensemble

```
"A simple Molecular Dynamics program for
Lennard-Jones particles (using LJ units).
It realizes the NVE ensemble, i.e. the
number of particles, N, the volume, V,
and the energy of the system, E, are
constant.";

"The program INIT is used the initialize
various quantities. It also sets up the
initial particle positions and assigns
the initial random velocities to the
particles";

"Individual key pieces: MOVER - advaces
the positions, MOVEV - advaces the
velocities, FORCE - calculates the
forces felt by the individual particles
at their current positions";

"INIT";
"set parameters:";
"number of particles - (too) small because we use
Mathematica!!";
n = 3 * 3 * 3;
"number of time steps";
NSTEP = 100000;
```

```
"maximum magnitude of initial random velocity component";
vmax = 2.7;
"cut-off radius for the forces";
rcut = 3;
"the primary simulation box volume is V=L^3";
L = 3;
"timestep";
Δt = 0.001;

"generate initial coordinates on a cubic lattice";
"initialize coordinate arrays";
x = Table[0, {i, 1, n}, {k, 1, NSTEP}];
y = Table[0, {i, 1, n}, {k, 1, NSTEP}];
z = Table[0, {i, 1, n}, {k, 1, NSTEP}];
"calculate particle coordinates on cubic lattice";
i = 0; max = n^(1/3); Do[
i+=1;
x[[i, 1]] = ii;
y[[i, 1]] = jj;
z[[i, 1]] = kk,
{ii, 0, max - 1},
{jj, 0, max - 1},
{kk, 0, max - 1}];
"display particles on cubic lattice and box boundaries";
g1 =
Graphics3D[
{Table[{PointSize[Large],
Point[{x[[i, 1]], y[[i, 1]], z[[i, 1]]}]}, {i, 1, n}],
Line[{{0, 0, 0}, {L, 0, 0}, {L, L, 0}, {0, L, 0}, {0, 0, 0},
```

```
{0, 0, L}, {L, 0, L}, {L, L, L}, {0, L, L}, {0, 0, L},
{0, L, L}, {0, L, 0}, {L, L, 0}, {L, L, L}, {L, 0, L},
{L, 0, 0}}]}, Boxed → False]


"generate random velocity components";
vx = Table[0, {i, 1, n}, {k, 1, NSTEP}];
vy = Table[0, {i, 1, n}, {k, 1, NSTEP}];
vz = Table[0, {i, 1, n}, {k, 1, NSTEP}];
Do[
vx[[i, 1]] = vmax(2Random[Real, 1] − 1);
vy[[i, 1]] = vmax(2Random[Real, 1] − 1);
vz[[i, 1]] = vmax(2Random[Real, 1] − 1),
{i, 1, n}];
"subtract center of mass velocity";
vxcm = Sum[vx[[i, 1]], {i, 1, n}]/n
vycm = Sum[vy[[i, 1]], {i, 1, n}]/n
vzcm = Sum[vz[[i, 1]], {i, 1, n}]/n
Do[
vx[[i, 1]] = vx[[i, 1]] − vxcm;
vy[[i, 1]] = vy[[i, 1]] − vycm;
vz[[i, 1]] = vz[[i, 1]] − vzcm,
{i, 1, n}];
"check this";
vxcm = Sum[vx[[i, 1]], {i, 1, n}]/n
vycm = Sum[vy[[i, 1]], {i, 1, n}]/n
vzcm = Sum[vz[[i, 1]], {i, 1, n}]/n
"initialize force array";
fx = Table[0, {i, 1, n}, {k, 1, NSTEP}];
fy = Table[0, {i, 1, n}, {k, 1, NSTEP}];
```

fz = Table[0, {i, 1, n}, {k, 1, NSTEP}];



0.480694

−0.11963

−0.193074

1.0691036533427433*^-16

−4.111937128241328*^-17

0.

"NVE - MD for LJ particles";

Timing[

"FORCE (k) ";

k = 1;

Do[

xmin = (x[[i, k]] − x[[j, k]]) − L Round $\left[\frac{(x[[i,k]]-x[[j,k]])}{L}\right]$;

ymin = (y[[i, k]] − y[[j, k]]) − L Round $\left[\frac{(y[[i,k]]-y[[j,k]])}{L}\right]$;

zmin = (z[[i, k]] − z[[j, k]]) − L Round $\left[\frac{(z[[i,k]]-z[[j,k]])}{L}\right]$;

rmin2 = xmin^2 + ymin^2 + zmin^2;

If[rmin2 < rcut^2,

{f = 48/rmin2^7 − 24/rmin2^4;

fx[[i, k]]+= f xmin;

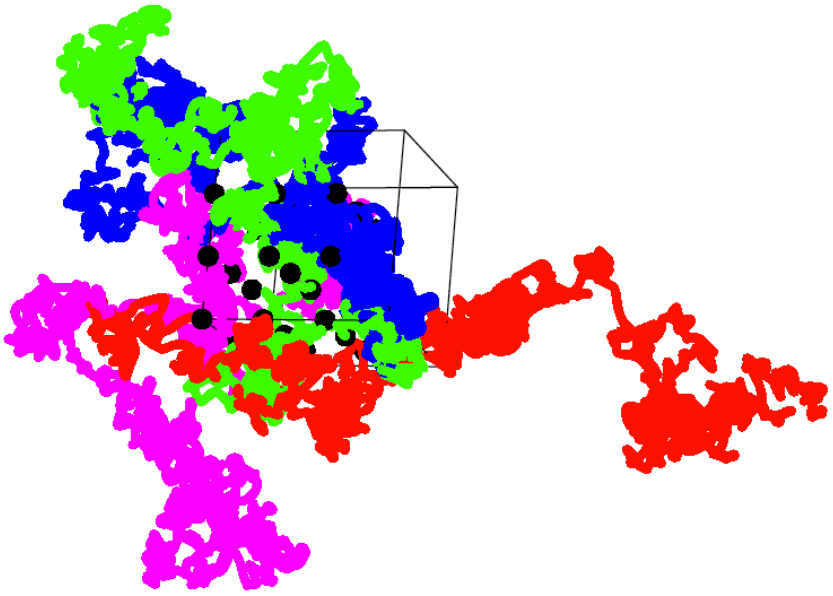
```
fy[[i, k]] += f ymin;
fz[[i, k]] += f zmin;
fx[[j, k]] += - f xmin;
fy[[j, k]] += - f ymin;
fz[[j, k]] += - f zmin}],
{i, 1, n - 1}, {j, i + 1, n}];

"main loop of MD";

Do[

"MOVER (k-1)";
Do[
x[[i, k]] = x[[i, k - 1]] + Δtvx[[i, k - 1]] +
(Δt^2/2)fx[[i, k - 1]];
y[[i, k]] = y[[i, k - 1]] + Δtvy[[i, k - 1]] +
(Δt^2/2)fy[[i, k - 1]];
z[[i, k]] = z[[i, k - 1]] + Δtvz[[i, k - 1]] +
(Δt^2/2)fz[[i, k - 1]],
{i, 1, n}];

"FORCE (k) ";
Do[
xmin = (x[[i, k]] - x[[j, k]]) - L Round [(x[[i,k]] - x[[j,k]]) / L];
ymin = (y[[i, k]] - y[[j, k]]) - L Round [(y[[i,k]] - y[[j,k]]) / L];
zmin = (z[[i, k]] - z[[j, k]]) - L Round [(z[[i,k]] - z[[j,k]]) / L];
rmin2 = xmin^2 + ymin^2 + zmin^2;
If[rmin2 < rcut^2,
{f = 48/rmin2^7 - 24/rmin2^4;
```

```
fx[[i, k]] += fxmin;
fy[[i, k]] += fymin;
fz[[i, k]] += fzmin;
fx[[j, k]] += - fxmin;
fy[[j, k]] += - fymin;
fz[[j, k]] += - fzmin}],
{i, 1, n - 1}, {j, i + 1, n}];

"MOVEV (k-1)";
Do[
vx[[i, k]] = vx[[i, k - 1]] + (Δt/2)(fx[[i, k]] + fx[[i, k - 1]]);
vy[[i, k]] = vy[[i, k - 1]] + (Δt/2)(fy[[i, k]] + fy[[i, k - 1]]);
vz[[i, k]] = vz[[i, k - 1]] + (Δt/2)(fz[[i, k]] + fz[[i, k - 1]]),
{i, 1, n}],

{k, 2, NSTEP}]]
```

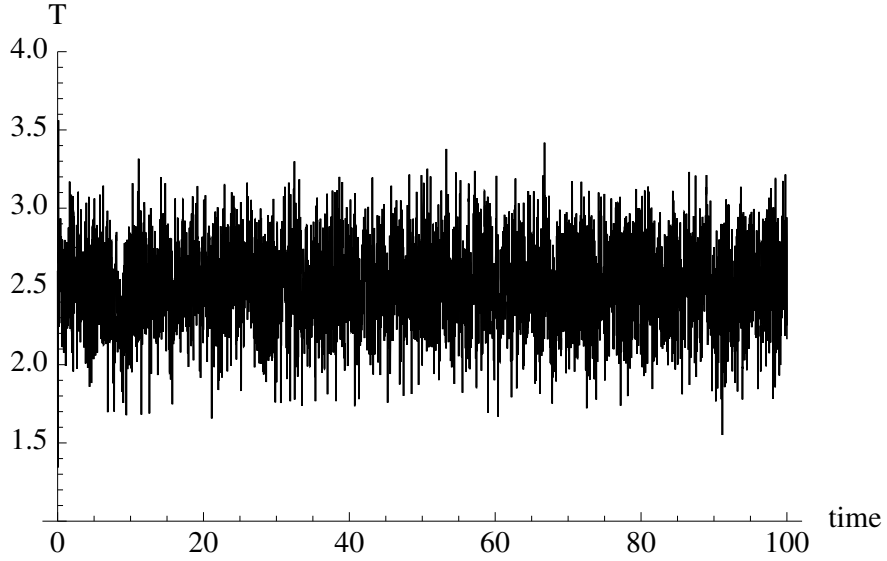
```
{1204.27, Null}
```

```
"pictorial representation of selected particlès path
including the initial lattice";
g2 =
Graphics3D[
{Red, Point[Table[{x[[1, k]], y[[1, k]], z[[1, k]]},
{k, 2, NSTEP}]],
{Green, Point[Table[{x[[8, k]], y[[8, k]], z[[8, k]]},
{k, 2, NSTEP}]]},
{Blue, Point[Table[{x[[16, k]], y[[16, k]], z[[16, k]]},
{k, 2, NSTEP}]]},
```

```
{Magenta, Point[Table[{x[[24, k]], y[[24, k]], z[[24, k]]},
{k, 2, NSTEP}]]}, Boxed → False];
Show[g1, g2]
```



```
"Instantaneous temperature vs. time";
Table[
{Δtk, Sum[(vx[[i, k]]^2 + vy[[i, k]]^2 + vz[[i, k]]^2), {i, 1, n}]/
(3n)}, {k, 2, NSTEP}];
ListPlot[%, Joined → True, PlotRange → {1, 4}, PlotStyle → Black,
AxesLabel → {"time", "T"}]
```



References

(1) R. Hentschke *Thermodynamics* Springer: Heidelberg, **2014**.

(2) R. Hentschke *Statistische Mechanik* Wiley-VCH: Weinheim, **2004**.

(3) R. Hentschke, E.M. Aydt, B. Fodi, E. Stöckelmann *Molekulares Modellieren mit Kraftfeldern.*

(4) Handbook of Chemistry and Physics, (Ed. D. R. Lide), CRC Press:Boca Raton

(5) R. Hentschke *A short Introduction to Quantum Theory*, lecture notes

(6) D. S. Evans, *Statistical Mechanics of Nonequilibrium Liquids* Cambridge University Press: Cambridge, **2014** (section 4); S. Engelmann, J. Meyer, R. Hentschke *Computer Simulation of Thermal Conductivity in vulcanized polyisoprene at variable strain and temperature* Physical Review B *96*, 054110 (**2017**)