Worksheet 1: Integrators

Marko Zecevic & Julia Hickl

November 23, 2023

Institute for Computational Physics, University of Stuttgart

Contents

1 Exercise 2: Statistical Mechanics

1.1 Statistical Mechanics

Task: Consider a System A that consists of subsystems A_1 and A_2, with numbers of possible configurations $\Omega_1 = 10^{31}$ and $\Omega_2 = 10^{28}$, respectively. What is the number of configurations available to the combined system? Also, compute the entropies S, S_1 and S_2.

Solution: The number of configurations available to the combined system is $\Omega = \Omega_1 \cdot \Omega_2$. The entropy can be calculated from the number of possible configurations with

$$S = k_\mathrm{B} \ln(\Omega). \tag{1}$$

Therefore, the entropies of the two individual systems and the combined system can be calculated to

$$\begin{aligned} S &= \ln(\Omega_1 \cdot \Omega_2) \cdot k_\mathrm{B} = 59 \cdot k_\mathrm{B} \cdot \ln(10) = 1.876 \cdot 10^{-21} \frac{\mathrm{J}}{\mathrm{K}} \\ S_1 &= 31 \cdot k_\mathrm{B} \cdot \ln(10) = 9.855 \cdot 10^{-22} \frac{\mathrm{J}}{\mathrm{K}} \\ S_2 &= 28 \cdot k_\mathrm{B} \cdot \ln(10) = 8.901 \cdot 10^{-22} \frac{\mathrm{J}}{\mathrm{K}}. \end{aligned}$$

Task: By what factor does the number of available configurations when 1 m^3 of neon at 1 atm and 298 K is allowed to expand by 1% at constant temperature? (Neon should be treated as an ideal gas here.)

Solution: The change of available configurations can be calculated by dividing the available configurations after the expansions by the available configurations before the expansion

$$\frac{\Omega_\mathrm{a}}{\Omega_\mathrm{b}} = \exp\left(\frac{1}{k_\mathrm{B}}(S_\mathrm{a} - S_\mathrm{b})\right) = \exp\left(\frac{1}{k_\mathrm{B}}\Delta S\right). \tag{2}$$

For an isothermal expansion, the difference in entropy can be calculated with

$$\Delta S = nR \ln\left(\frac{V_\mathrm{a}}{V_\mathrm{b}}\right), \tag{3}$$

where n is the number of particles, R is the gas constant and V_a and V_b are the volumes after and before the expansion. With the ideal gas

$$pV = nRT \Longleftrightarrow nR = \frac{pV}{T} \tag{4}$$

the entropy difference becomes

$$\Delta S = \frac{pV}{T} \ln\left(\frac{V_\mathrm{a}}{V_\mathrm{b}}\right) = \frac{101\,325\,\mathrm{Pa}\cdot 1\,\mathrm{m}^3}{298\,\mathrm{K}} \cdot \ln(1.01) = 3.383\frac{\mathrm{J}}{\mathrm{K}}.$$

With this result, we obtain an increase in available configurations

$$\frac{\Omega_\mathrm{a}}{\Omega_\mathrm{b}} = \exp\left(\frac{3.383}{1.380\cdot 10^{-23}}\right) = 10^{10^{25}}.$$

Task: By what factor does the number of configurations increase when an energy of 100 kJ is added to a system at initially $T_0 = 400\mathrm{K}$ containing 1.0 mol of particles at constant volume?

Solution: For heating with constant volume the change in entropy is

$$\Delta S = nC_\mathrm{v} \ln\left(\frac{T}{T_0}\right), \tag{5}$$

where C_v is the heat capacity

$$C_\mathrm{v} = \frac{nR}{\gamma - 1}, \qquad \gamma = 1 + \frac{2}{f} \tag{6}$$

with f being the degrees of freedom. For each particle in the gas $f = 3$, therefore $\gamma = 5/3$. Since energy is added to our system, the first law of thermodynamics

$$\mathrm{d}U = \mathrm{d}Q + \mathrm{d}W \tag{7}$$

can be applied. Since $V = \text{const.}$, $\mathrm{d}W = 0$. Therefore we obtain

$$\mathrm{d}U = \mathrm{d}Q = \frac{3}{2}nR\Delta T \Longleftrightarrow \Delta T = \frac{2}{3}\frac{\Delta Q}{nR}$$

and can plug this result in (5), from which we then obtain

$$\Delta S = n\frac{3}{2}nR \ln\left(\frac{\frac{2}{3}\frac{\Delta Q}{nR} + 400}{400}\right) = 2.288\cdot 10^{25}\frac{\mathrm{J}}{\mathrm{mol}\cdot\mathrm{K}}.$$

The change of available configurations then increases by

$$\frac{\Omega_\mathrm{a}}{\Omega_\mathrm{b}} = \exp\left(\frac{\Delta S}{k_\mathrm{B}}\right) = 10^{10^{47.86}}$$

Task (Ideal Gas): The canonical partition function of an ideal gas consisting of N mono-atomic particles is

$$Z(N,V,T) = \frac{1}{h^{3N}N!}\int \mathrm{d}\Gamma \exp\left(-\beta H\right) \tag{8}$$

in which $\beta = 1/(k_\mathrm{B}T)$ is the inverse temperature, the Hamiltonian is given by

$$H = \sum_{i=1}^{N}\frac{\mathbf{p}_i^2}{2m} \tag{9}$$

and $\mathrm{d}\Gamma = \mathrm{d}q_1...\mathrm{d}q_N\mathrm{d}p_1...\mathrm{d}p_N$. Derive expressions for the following thermodynamic properties:

- $F(N,V,T)$ (hint: $\ln N! \approx N\ln\left(N\right) - N$)
- C_V (heat capacity at constant volume) and p (pressure)

Solution: The free energy is caluclated with

$$F = -\frac{1}{\beta}\ln\left(Z\right)$$

therefore we need to calculate the canonical partition function

$$Z = \frac{1}{h^{3N}N!}\int \mathrm{d}\Gamma \exp\left(-\beta\sum_{i=1}^{N}\frac{\mathbf{p}_i^2}{2m}\right) = \frac{1}{h^{3N}N!}\left(\frac{2\pi m}{\beta}\right)^{3N/2}V^N = \frac{1}{N!}\frac{V^N}{\lambda^{3N}},$$

with $\lambda = h\sqrt{\frac{\beta}{2\pi m}}$. Plugging this into equation 10 leads to

$$\begin{aligned} F &= -\frac{1}{\beta}\ln\left(\frac{1}{N!}V^N(\lambda)^{-3N}\right) = -\frac{1}{\beta}\left(-\ln(N!) + N\ln V - N\ln\left(\lambda^3\right)\right. \\ &\approx -\frac{1}{\beta}\left(-N\ln(N) + N + N\ln V - N\ln\left(\lambda^3\right) = \frac{N}{\beta}\left(\ln\left(\frac{N\lambda^3}{V}\right) - 1\right).\right. \end{aligned}$$

The energy of the system is then given by

$$\begin{aligned} U &= \partial_\beta(\beta F) = \partial\beta N\left(\ln\left(\frac{N}{V}\left(\frac{h}{\sqrt{2\pi m}}\sqrt{\beta}\right)^3 - 1\right)\right) \\ &= \frac{3}{2}Nk_\mathrm{B}T, \end{aligned}$$

which leads to a heat capacity of

$$C_\mathrm{V} = \left(\frac{\mathrm{d}U}{\mathrm{d}T}\right) = \frac{3}{2}Nk_\mathrm{B}.$$

For the pressure of the system we obtain

$$p = -\partial_\mathrm{V}F = -\partial_\mathrm{V}\frac{N}{V}\left(\ln\left(\frac{N\lambda^3}{V}\right) -\right) = \frac{Nk_\mathrm{B}T}{V}.$$

1.2 Ising Model

In this section, we will regard a system of N spins in a general d-dimensional lattice with periodic boundary conditions (PBC). This means, that the spin on the last lattice site s_n not only interacts with its direct neighbor s_{n-1} but also with the first spin s_1.

Energy of the Ising model

The energy of the system can generally be calculated with

$$U = -H \sum_{i=1}^{N} s_i - J \sum_{\langle i,j \rangle} s_i s_j \tag{10}$$

where H describes the present magnetic field and J is the coupling between neighboring spins. The second summation is performed over all neighboring spin pairs in the system.

For the case of no external magnetic field, $H = 0$, we can compute the energy of the Ising model. Additionally, we will assume the system to be in its ground state, i.e. a temperature of $T = 0$. In this state, all spins align and the spin interaction simplifies to $s_i s_j = 1$ for both possible spin orientations. Therefore, the energy equation can be written as

$$U = -J \sum_{\langle i,j \rangle} 1. \tag{11}$$

The number of nearest neighbors every spin has in a d-dimensional grid with PBC is $2d$. This can be explained by introducing lattice vectors $\hat{e}_i$ for all dimensions. For every available dimension, it is possible to find the nearest neighbors by moving by one lattice vector in the grid. Due to the PBC, this is possible for all spins, therefore resulting in $2d$ neighbors per spin. However, this would lead to the consideration of every spin pair twice. Therefore, it is necessary to only consider spin pairs in the positive direction of the lattice vectors, leading to d significant neighbors per spin.
Finally, by summing over all N spins we end up with the energy

$$U = -J \sum_{i=1}^{N} d = -dNJ \tag{12}$$

of the Ising model in its ground state.

Derivation of free Energy (in 1d)

Next, we are interested in calculating the free energy per spin

$$\frac{F(\beta, N)}{N} = -\frac{1}{N\beta} \ln(Z(\beta)) \tag{13}$$

with $Z(\beta)$ being the partition function. This, in turn, can be computed by

$$Z(\beta) = \sum_{\alpha} \exp -\beta E_{\alpha} = \sum_{\{s_i=\pm 1\}} \exp(\beta J \sum_{i=1}^{N} s_i s_j). \tag{14}$$

Instead of summing over all spins configuration, one can introduce the new variable $\mu_i = s_i \cdot s_{i+1}$ as the coupling between two neighboring spins. This variable can only hold the values 1 and -1 for the cases $s_i = s_{i+1}$ and $s_i = -s_{i+1}$, where both spin configurations can be realized with two settings. Therefore, equation 14 can be rewritten as

$$Z(\beta) = 2 \sum_{\mu_i=\pm 1} exp(\beta J \sum_{i=1}^{N-1} \mu_i) = 2 \sum_{\mu_1=\pm 1} e^{\beta J \mu_1} \sum_{\mu_2=\pm 1} \cdots \sum_{\mu_{N-1}=\pm 1} e^{\beta J \mu_{N-1}}. \tag{15}$$

Next, it is possible to use the definition of the hyperbolic cosine $\cosh x = \frac{\exp(x)+\exp(-x)}{2}$ to simplify all sums, resulting in

$$Z(\beta) = 2^N cosh(\beta J)^{N-1} \quad \xrightarrow{N\to\infty} \quad (2\, cosh(\beta J))^N . \tag{16}$$

Finally, the free energy per spin can be computed in the limits of $N \to \infty$ with 13, which results in

$$\frac{F(\beta, N)}{N} = -\frac{1}{N\beta} \ln \left[(2 \cosh(\beta J))^N \right] = -\frac{2 \cosh(\beta J)}{\beta}. \tag{17}$$

Computation of the Energy and heat capacity

With the now known partition function $Z(\beta)$, we can derive expressions for the mean energy $\langle E \rangle$ and the heat capacity C_V.
The former is defined as

$$\langle E \rangle = -\frac{1}{Z_N(\beta)} \partial_\beta Z_N(\beta) = -\partial_\beta \ln(Z(\beta)), \tag{18}$$

in the canonical ensemble, while the latter can be computed from the energy as

$$C_V = \partial_T \langle E \rangle = -k_B \beta^2 \partial_\beta \langle E \rangle. \tag{19}$$

For the one dimensional Ising model, the energy can be written as

$$\langle E \rangle = -2N \partial_\beta \cosh(\beta J) = -2JN \sinh(\beta J) \tag{20}$$

and the heat capacity as

$$C_V = -k_B \beta^2 \partial_\beta \left(-2NJ \sinh(\beta J) \right) = 2 k_B N (\beta J)^2 \cosh(\beta J). \tag{21}$$

2 Exercise 3: Lennard-Jones Fluid

The complete code that was used to in the following tasks can be seen in the github repository `https://github.com/Marcevko/SimMethExercises.git` in the folder `sm1_worksheet_2`.

2.1 Lennard-Jones Potential

Task: Implement the Lennard-Jones potential and force in reduced units ($\varepsilon = \sigma = 1$) for two particles with the distance vector $\mathbf{r}_{ij}$.

Solution: The general Lennard-Jones potential is given by

$$V_{LJ}(r) = 4\varepsilon \left(\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right) \tag{22}$$

with the tuning parameters ε and σ. The acting forces in the potential is given its negative gradient, which results in

$$F_{LJ}(\mathbf{r}) = -\nabla_r V_{LJ} = 24\varepsilon \cdot \frac{\mathbf{r}}{r^2} \left(\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right). \tag{23}$$

However, in general, it is possible to set these parameters to $\varepsilon = \sigma = 1.0$ and transfer all simulated results to parameter sets of interest. This is the so-called `theorem of corresponding states`. Therefore, we will only implement the potential as well as the force independent of the parameters.

The implementation of the potential and the force can be found in the following code segment with the functions `lj_potential` and `lj_force`, respectively.

```python
import numpy as np

def lj_potential(r_ij: np.ndarray) -> float:
    vector_norm = np.linalg.norm(r_ij)
    return 4 * ( (1 / vector_norm**12) - (1 / vector_norm**6) )

def lj_force(r_ij: np.ndarray) -> np.ndarray:
    vector_norm = np.linalg.norm(r_ij)
    return 24 * r_ij * ( (2 / vector_norm**14) - (1 / vector_norm**8) )
```

Next, we plotted these functions for 2d-Numpy-arrays $(d, 0)$ with $d \in [0.85, 2.5]$. The resulting plots for the potential and the force (x-component) can be seen in Figure 1 in blue and orange.

Additionally, we plotted a shifted version of the Lennard-Jones potential, that is fully repulsive. To be able to obtain this potential, one has to shift the conventional potential by the minimum $V_{LJ}(r_{min} = 2^{1/6}) = -1$ and truncate it beyond that point. The code that was used to generate all three curves is given below.

```python
distance_vector = np.zeros((1000, 2))
distance_vector[:, 0] = np.linspace(0.85, 2.5, 1000)

lj_potential_computed = np.array(
    [lj_potential(distance) for distance in distance_vector]
)
lj_force_computed = np.array(
    [lj_force(distance) for distance in distance_vector]
)
lj_potential_cutoff = np.array(
    [lj_potential(distance) + 1 if np.linalg.norm(distance) <= \
    np.power(2, 1/6) else 0.0 for distance in distance_vector]
)
```

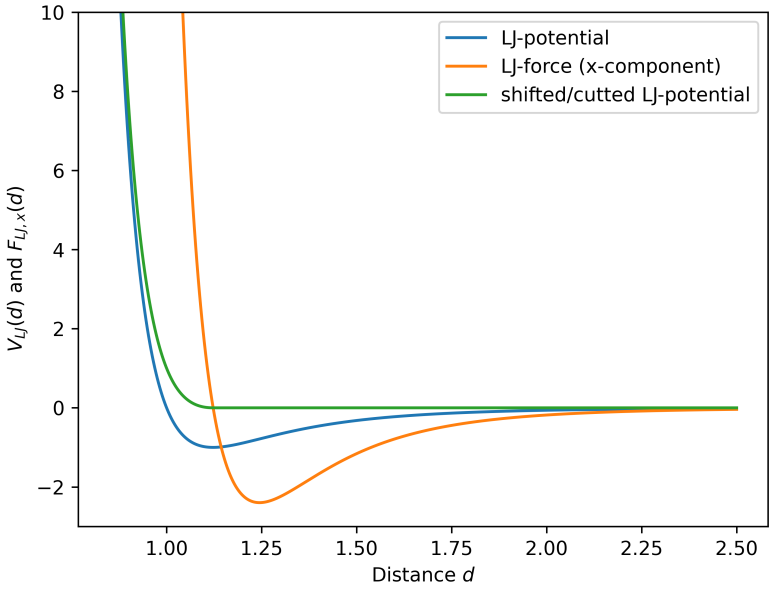


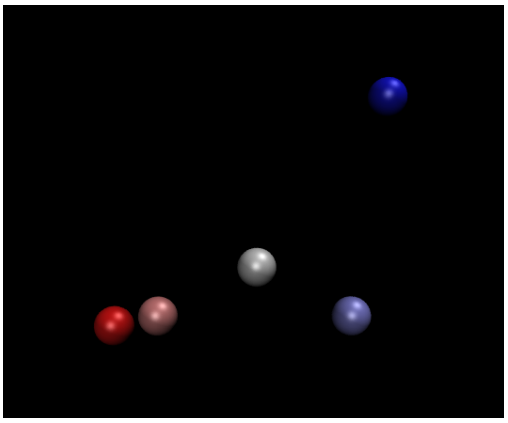
Figure 1: Plots of the Lennard-Jones potential and force as given in equations 22 and 23 blue and orange. Additionally, a fully repulsive version of the Lennard-Jones potential was shifted by 1 and cut after $r_{min} = 2^{1/6}$ in green.

2.2 Lennard-Jones Billiards

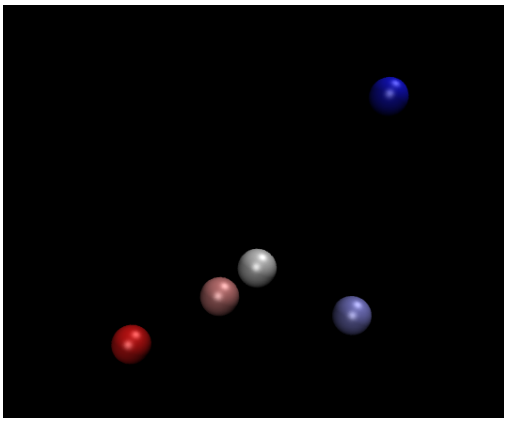
Task: Let the program run and visualize the system with VMD.

Solution: In the file `ex_3_3.py`, we were given an already implemented code that runs a Lennard-Jones Billiard system with the in section 2.1 discussed Lennard-Jones potential and force. When run, the script writes a `lj_billiard.vtf`-file containing the trajectories of all particles. These trajectories can then be visualized with the software `VMD`. We created four screenshots of the simulated dynamics, which can be found in figure 2.

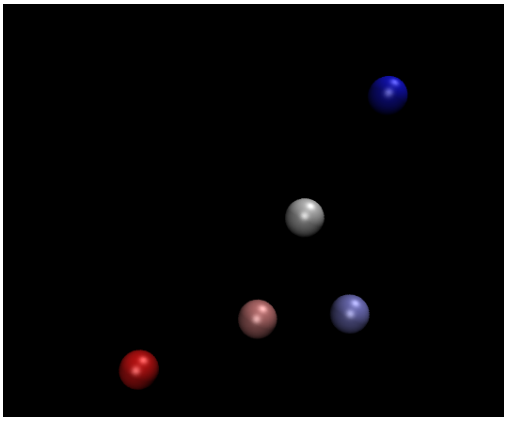
The first image in subfigure 2a shows the system short after the initialization. The red particle is flying towards the coral red colored one along an imaginary x-axis, while all other particles rest. Therefore, the red atom is about the hit the coral red colored particle, starting the "billiard game".



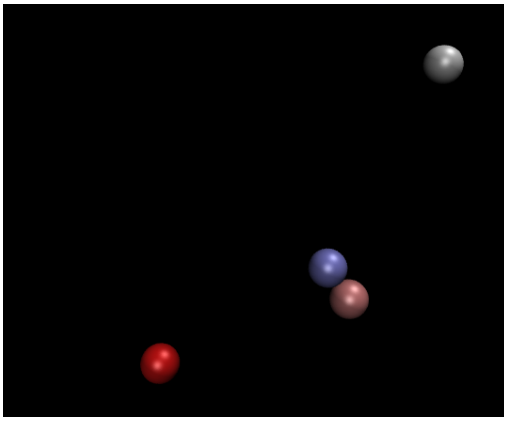
(a) Before the first impact



(b) After the first impact



(c) After the second impact



(d) Particles 2 & 4 attracting each other

Figure 2: Rendering in VMD of the Lennard-Jones Billiard system at four different time points.

The second shot in subfigure 2b shows the system after the first impact. The red sphere seems to move slowly towards the bottom right, while the coral-colored sphere is about to hit the silver-colored sphere. Both atoms that were involved in the first collision seem to behave like classic billiard balls.

The third screenshot in 2c displays the system after the second collision. The coral atom is now moving in the vicinity of the lower blue sphere. Moreover, the silver ball is now moving toward the upper blue atom in a straight line.

The last shot shows the system after the silver and upper blue particles have collided. What is of more interest though, are the changed trajectories of the coral and blue atoms. Apparently, they have not moved in straight lines or rested in the meantime, as would be expected of classic billiard balls, but rather seem to be attracting each other rotating around their center of mass.

Task: Change the position of the fifth particle (index 4) so that it is hit by the third particle (index 2).

Solution: In figure 3 the trajectories of the five particles can be seen, as well as the total energy. From the figure, it becomes apparent that the third particle already hits the fifth particle, so no further changes were made. It can also be seen that the energy of the system is conserved, as is expected with the velocity verlet integrator.

Task: Although the particles almost behave like billiard balls, some of them do not. Which particles and why? Create a plot of trajectory or a series of images that supports your hypothesis.

Solution: In figure 3 it can already be seen that the second particle (index 1) and the fourth particle (index 3) do not behave like billiard balls since they cross paths and do not have a straight trajectory after the collision. Zooming in on the collision point of both particles, as can be seen in figure 4, confirms this. The particles do not behave like the other particles, since they do not actually collide. Instead, they first attract each other, which leads to the crossing of their trajectories.

Task: Now introduce 2D boundaries to confine the system. Extend ex_3_3.py in order to add four hard walls. They should be placed at $x = 0$, $x = \text{box1}$, $y = 0$ and $y = \text{box1}$, with $\text{box1} = 15$. The interaction of the particles with the walls should be elastic. Create a plot of the trajectory. Is the energy of the system sill conserved?

Solution: When the particles elastically collide with the wall, the direction of the velocity changes, while the absolute values stays the same. If the particle collides with a wall parallel to the y-direction, the velocity in the y-direction stays the same, while the x-component of the velocity is flipped by a minus. If the particle collides with a wall parallel to the x-direction, the y-component of the velocity flips. The implementation in python can be seen in the following code.

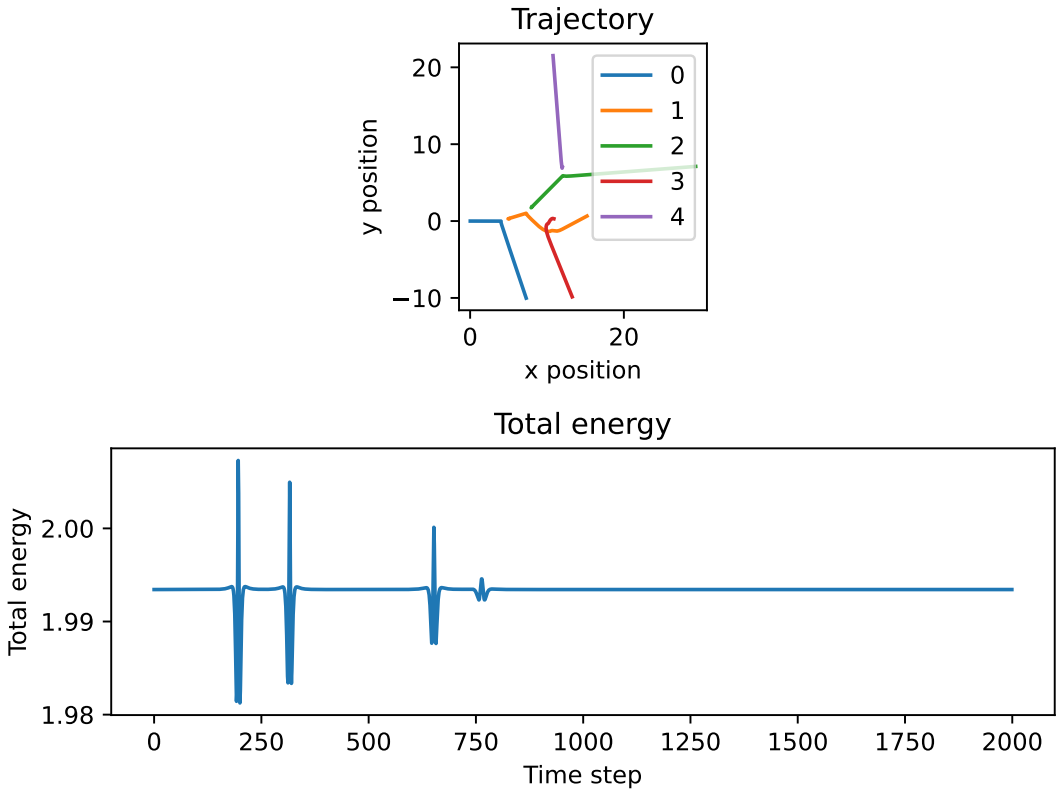


Figure 3: Simulated billiard balls without boundary conditions.

```
for i in range(len(x[0, :])):
    if x[0, i] < 0 or x[0, i] > box_length:
        v[0, i] = - v[0, i]
    if x[1, i] < 0 or x[1, i] > box_length:
        v[1, i] = - v[1, i]
```

When simulating the five particles with these boundaries, the trajectories seen in figure 5 are obtained. It can be seen that the particles show the expected behavior when colliding with the wall. Since the particles collide elastically, the energy of the system should be conserved. However, as can be seen in the energy diagram, during the first collision the energy of the system is reduced. The reason for that is that the starting point of the first particle is at [0, 0], with a starting velocity [2.0, 0.0]. Therefore, the trajectory before the collision is right along the boundary, so the particle is not inside the box. To correct this, the simulation is run again, but this time the starting points of all particles are increased by 1 in both the x- and y-direction, so all particles in the box. The resulting trajectories are shown in figure 6, where it can be seen that the energy is now conserved.

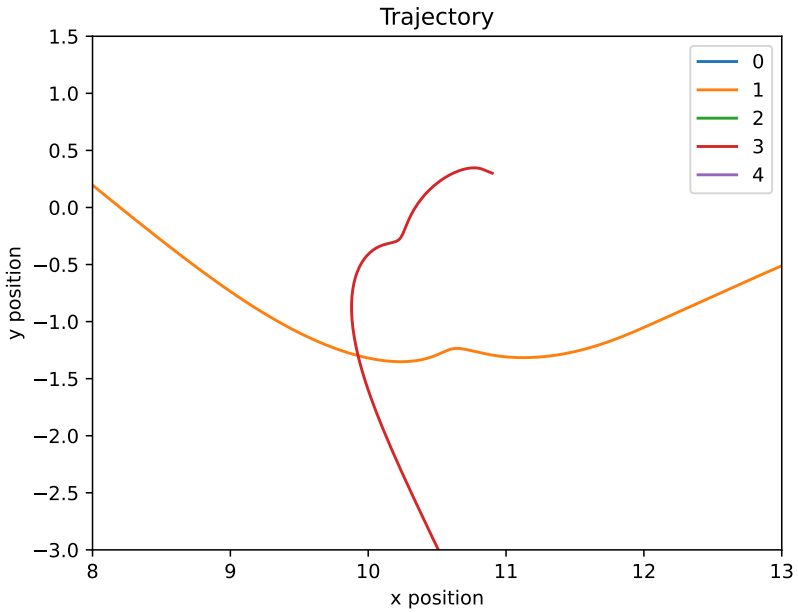


Figure 4: Trajectory of second and fourth particle, where it can be seen that they do not behave like billiard balls.

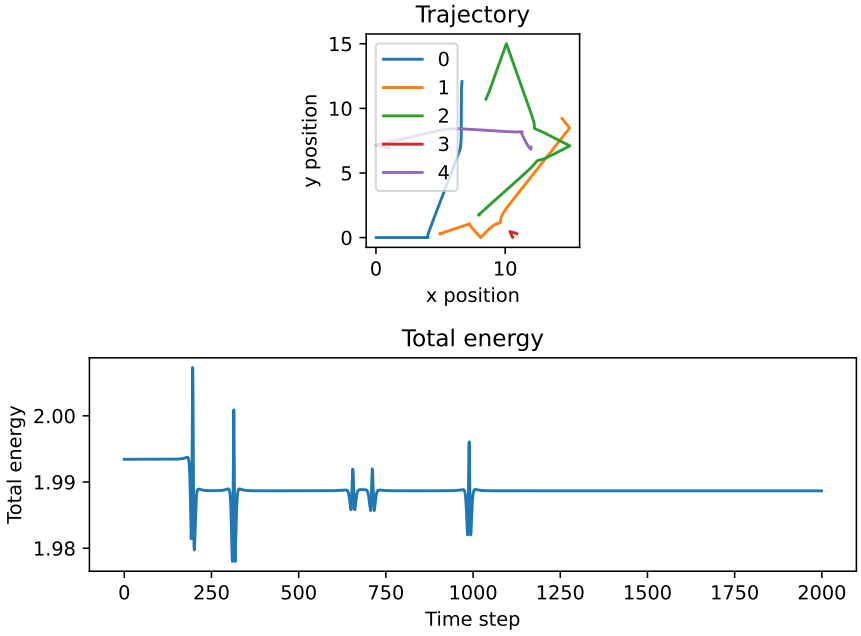


Figure 5: Simulated trajectories and energy of the Lennard-Jones Billiard system with hard walls.

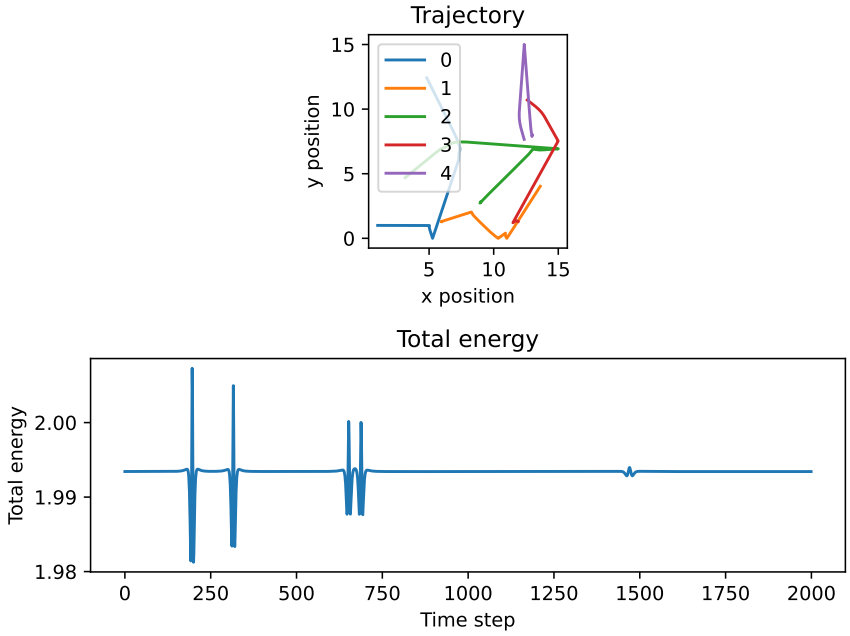


Figure 6: Simulated trajectories of the five particles where it was made sure that all particles were inside the box at the beginning of the simulation. As it can be seen, the energy of the system is now conserved.

2.3 Periodic Boundary Conditions

Task: Make a copy of the solution from the previous task. Modify the LJ potential and force such that it implements the truncated LJ potential with $R_{\text{cutoff}} = 2.5$.

Solution: To reduce computing time, a truncated LJ potential is introduced. This potential only considers interactions if the distance between two particles is smaller than a distance r_{cutoff}. To make sure the potential is continuous, the potential must be shifted. This leads to the truncated potential

$$V'_{\text{LJ}}(r) = \begin{cases} V_{\text{LJ}}(r) - V_{\text{LJ}}(r_{\text{cutoff}}) & r \leq r_{\text{cutoff}} \\ 0 & r > r_{\text{cutoff}} \end{cases}$$

with the corresponding force

$$F'_{\text{LJ}}(r) = \begin{cases} F_{\text{LJ}}(r) & r \leq r_{\text{cutoff}} \\ 0 & r > r_{\text{cutoff}}. \end{cases}$$

The implementation to python can be seen in the following code.

```
R_CUTOFF = 2.5
SHIFT = - ex_3_2.lj_potential(R_CUTOFF)

def lj_force(r_ij: np.ndarray, r_cutoff: float) -> np.ndarray:
    vector_norm = np.linalg.norm(r_ij)
    return ex_3_2.lj_force(r_ij) if vector_norm <= r_cutoff else
    ↪  np.zeros(len(r_ij))

def lj_potential(r_ij: np.ndarray, r_cutoff: float, shift: float) ->
↪  float:
    vector_norm = np.linalg.norm(r_ij)
    return ex_3_2.lj_potential(r_ij) + shift if vector_norm <= r_cutoff
    ↪  else 0.0
```

Task: Extend the program such that it takes into account periodic boundary conditions. Which functions need to be modified to implement them?

Solution: To make sure the particles do not escape our box, we introduce periodic boundary conditions. To implement them to the code, the function that calculates the velocity verlet steps needs to be modified, so that if a particle leaves the box on one side, it in reintroduced on the opposite side of the box. This is implemented with the following code.

```
def step_vv(x: np.ndarray, v: np.ndarray, f: np.ndarray, dt: float,
↪  r_cutoff: float, box_length:float, box: bool = True):
    x += v * dt + 0.5 * f * dt * dt
```

```
    v += 0.5 * f * dt

    # apply PBC
    x = x % box_length

    f = forces(x, r_cutoff, box_length)
    v += 0.5 * f * dt

    return x, v, f
```

In addition to that, the calculation of the force needs to be modified. Since we consider periodic boundary conditions (PBC), the distance between two particles cannot simply be calculated by subtracting the position of one particle by the position of the other particle. If the distance in x- or y- direction is larger than half of the boxes length, the particles actually have a shorter distance due to the PBC. To implement this, a new function is introduced, that is used to calculated the distance between two particles for the force (and later potential) calculation.

```
def minimum_image_vector(x1: np.ndarray, x2: np.ndarray, box_length:
↪  float) -> np.ndarray:
    x_12 = x1 - x2
    interaction_distance = x_12 - box_length * np.round(x_12 / box_length)
    return interaction_distance
```

Task: To test the PBC, set up two particles in a system with edge length $L = 10.0$, initial positions $\mathbf{x}_0 = (3.9, 3)$ and $\mathbf{x}_1 = (6.1, 5)$, and initial velocities $\mathbf{v}_0 = (-2, -2)$ and $\mathbf{x}_2 = (2, 2)$. Simulate them for 20 time units. With PBC, the particles should collide. Plot the trajectory.

Solution: The simulated trajectory of the two particles in the system with the given initial conditions can be seen in figure 7. As can be seen, the PBC were implemented successfully and the particles do collide.

2.4 Lennard-Jones Fluid

Task: Extend the program to set up the particles on a cubic 2d-lattice with a given number of particles per dimension n at a given density ρ.

Solution: The implementation of the 2d-cubic lattice can be seen in the following code.

```
VOLUME = N_PART / DENSITY
PARTICLE_DISTANCE = np.sqrt(VOLUME)

x = np.zeros((2, N_PART))

```

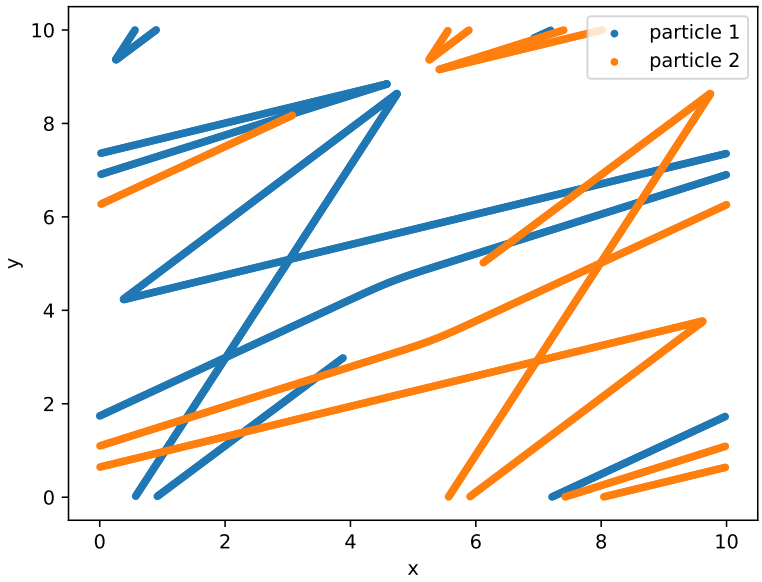


Figure 7: Simulated trajectories of two particles with implemented boundary conditions.

```
n = 0
for i in range(N_PER_SIDE):
    for j in range(N_PER_SIDE):
        x[0, test] += PARTICLE_DISTANCE*j + 1
        x[1, test] += PARTICLE_DISTANCE*i + 1
        n += 1
```

Task: Perform simulations of the LJ fluid with $n = \{3, 4, 5, ..., 12, 13\}$ particles per dimension of the quadratic lattice (time step $\Delta t = 0.01$, end time $t_{\max} = 1.0$, density $\rho = 0.7$) and measure their run times. Plot the measured run times over the respective total particle number. Produce not only a linear plot but also semilog and loglog plots. On the basis of these plots discuss the general scaling behavior. Based on your insights on the general scaling choose an appropriate fit function and explain your reasoning. Fit this function to your data and add the fit to your plots. In your report, give both the functional form and values of the fit parameters. Discuss the run time scaling behavior based on the results of your fit.

Solution: In figure 8 the results can be seen, where the runtime is plotted over the particle number in a linear, semi-logarithmic, and double-logarithmic plot.

Looking at the linear plot, it seems as though the computation time scales quadratically with the number of particles. Since the double-logarithmic plot shows a seemingly

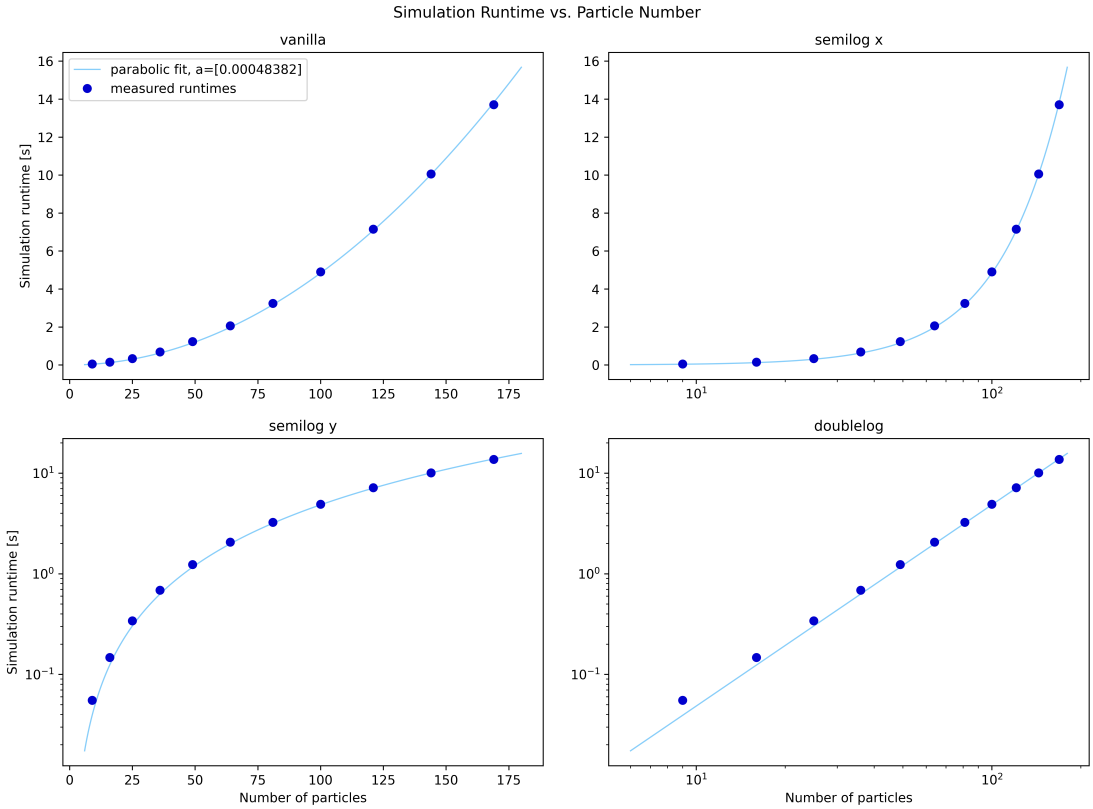


Figure 8: Computation time plotted over the total number of particles in a linear, semi-logarithmic and double-logarithmic plot. From the double-log plot it becomes apparent that the scaling should be polynomial, the linear plot suggests quadratic behavior. This is confirmed by the plotted quadratic fit function.

linear curve, a polynomial behavior is expected. To confirm this, a fit function of

$$f(x) = a \cdot x^b \tag{24}$$

is fitted to the values. With the fitting parameters this equation becomes

$$f(x) = 0.00189 \cdot x^{1.910},$$

which confirms the quadratic scaling quite well. When calculating the force, each particle interacts with the remaining particles. In total, $N(N-1)/2$ interactions need to be calculated, which is the reason for the quadratic scaling.

2.5 Verlet lists

Task: In the program the actual creation of the Verlet lists is missing. Implement the Verlet list creation function.

Solution: Verlet lists are implemented to reduce the computation time that scales with N^2. In order to do so,

```python
def get_verlet_list(x: np.ndarray, r_cut: float, skin: float, box:
↪  np.ndarray) -> (np.ndarray, np.ndarray):

N = x.shape[1]
verlet_list = []

for i in range(N):
    for j in range(i+1, N):
        r_ij = ex_3_4.minimum_image_vector(x[:, i], x[:, j], box)
        if np.linalg.norm(r_ij) < (r_cut + skin):
            verlet_list.append([i, j])

return np.copy(x), np.array(verlet_list)
```

Task: Why does the function return a copy of the current particle position in addition to the actual Verlet list?

Solution: If a particle moves out of the skin radius, the verlet lists need to be updated. In order to check for that, the distance between the initial particle positions and the current particle positions is calculated. If this distance is greater than the skin radius, a new verlet list is calculated.

Task: Measure the run time of the integration loop for the constant number of particles per dimension $n = 8$ and varying skin values skin $= \{0.0, 0.1, 0.2, \cdots, 0.9\}$.

Solution: The implementation of the Verlet-Lists was now combined with the given `ex_3_6.py` script and executed with the parameter $T_{MAX} = 40.0$. The resulting run times as well as the counted number of Verlet-List updates are visualized in figure 9.

The left-hand side of the figure shows a decrease in list updates with increasing skin values. This behavior is as intended, due to the update condition being defined over the thickness of the skin. The particles have to travel a larger distance to be able to cross half the skin and cause a verlet list update.

Next, the measured run times of the simulation are shown on the right side of the figure. Initially, with rising skin values, the run times drop significantly. This is caused combined effects of the Verlet lists and the skin value. As previously seen, increasing skin values lead to fewer list updates, which are scaling with N^2. Therefore, less updates can speed up the code. Moreover, with the introduction of the lists, we can neglect most of the force calculations that are also scaling with N^2. However, for further increasing skin values, the run times are again rising and leading to a minimum of around $skin = 0.3$. Due to the big skin values, we are now considering too many particles during the force calculation, which is making the algorithm less efficient.

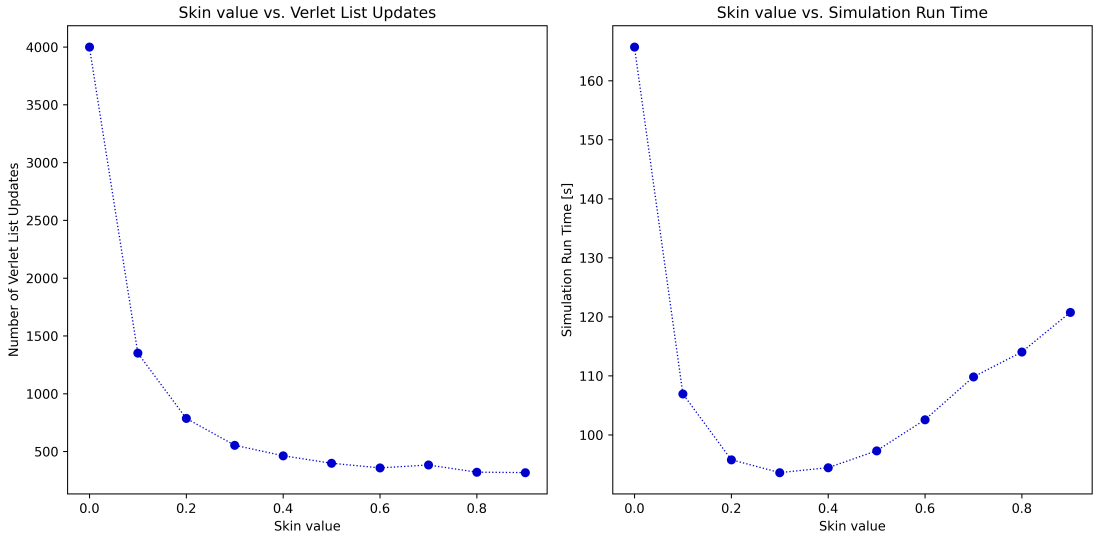


Figure 9: Plots of the number of Verlet-List updates during the simulations and their respective run times dependent on the skin value that were chosen for the Verlet-Lists.

Task: Choose the skin with the smallest run time from the previous task. Adapt the simulation time T_{MAX} to match the one used in task 3.5. and repeat the run time measurements of task 3.5.

Solution: Lastly, we are comparing the run times of the Lennard-Jones liquid with and without the use of Verlet lists. This is done by setting the skin to the previously found sweet spot $skin = 0.3$ and tuning the number of particles n. The resulting plots can be found in figure 10 in linear and double-logarithmic axes.

As expected, the Verlet list algorithm is scaling better than the default code. This can be seen in the double-logarithmic plot, which shows a flatter incline compared to the default code, indicating a smaller exponent in scaling behavior. For small atom numbers, however, the creation and updates to the Verlet lists lead to larger run times.

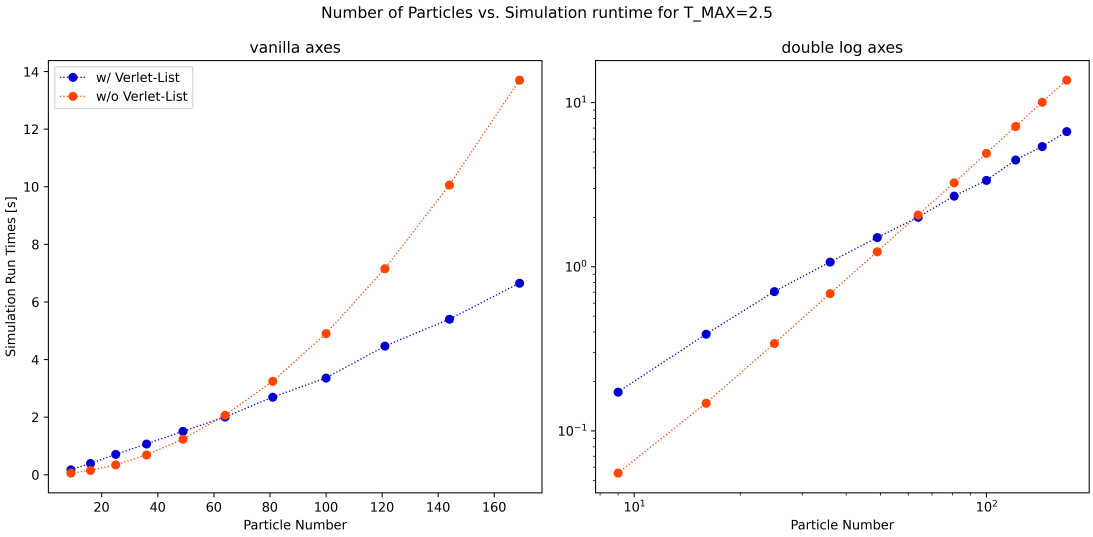


Figure 10: Comparison of the run times dependent on the particle number between systems with and without Verlet-lists.