Report for Worksheet 3: Molecular Dynamics 2 and Observables

Markus Baur and David Beyer

December 9, 2019

Contents

1 Saving and Restarting the Simulation

The state of a previous simulation is loaded in this fashion from the checkpoint file:

```
else:
    logging.info("Reading␣state␣from␣checkpoint.")
    with open(args.cpt, 'rb') as fp:
    state = pickle.load(fp)
    positions = state['positions']
    energies = state['energies']
    pressures = state['pressures']
    temperatures = state['temperatures']
    rdfs = state['rdfs']
    potential_energies = state['potential_energies']
    kinetic_energies = state['kinetic_energies']

    x = state['x']
    v = state['v']
    f = state['f']
    f_max = state['f_max']
```

During the simulation, these observables are measured:

```
for i in tqdm.tqdm(range(N_TIME_STEPS)):
    sim.propagate()

    if i % SAMPLING_STRIDE == 0:
        positions.append(sim.x.copy())
        pressures.append(sim.pressure())
        energies.append(np.sum(sim.energy()))
        temperatures.append(sim.temperature())
        rdfs.append(sim.rdf())
        potential_energies.append(sim.e_pot())
        kinetic_energies.append(sim.e_kin())
```

These observables are written to the checkpoint file in the end:

```
if args.cpt:
    state = {'positions': positions,
             'energies': energies,
             'pressures': pressures,
             'temperatures': temperatures,
             'rdfs': rdfs,
             'potential_energies': potential_energies,
             'kinetic_energies': kinetic_energies,
             'x':sim.x,
             'v': sim.v,
             'f': sim.f,
             'f_max': sim.f_max}
    write_checkpoint(state, args.cpt, overwrite=True)
```

2 Simple Observables

The kinetic energy is calculated from the velocities:

```
def e_kin(self):
    return np.sum(0.5 * np.power(self.v, 2))  # mass = 1
```

The potential energy can be calculated as the sum of the entries of the array e_pot_ij_matrix. Because every pair is summed over twice, a factor of 0.5 is needed:

```
def e_pot(self):
    return np.sum(self.e_pot_ij_matrix) / 2
```

The function energy returns a NumPy array which contains the potential and kinetic energy:

```
def energy(self):
    self.energies()

    return np.array((self.e_pot(), self.e_kin()))
```

The temperature of the system is calculated using the equipartition theorem, it is implemented in this way:

```
def temperature(self):
    return 2 * self.e_kin() / (self.n_dims * self.n)
```

The pressure is calculated using the formula given in the lecture/on the worksheet:

```
def pressure(self):
    f = np.multiply(self.f_ij_matrix, self.r_ij_matrix)
    f = np.sum(f)

    area = np.product(self.box)
    ret = 1 / area * (self.e_kin() + f * 0.25)
    return ret
```

The sum $\sum_{i<j}\langle \mathbf{r}_{ij} \cdot \mathbf{F}_{ij} \rangle$ is calculated as a pointwise multiplication and subsequent summation of the arrays f_ij_matrix and r_ij_matrix. Because the sum only runs over $i < j$, an additional factor of 0.5 is needed.

3 Equilibration

Figure 1 shows the time evolution the potential energy E_{pot}, kinetic energy E_{kin}, pressure P and temperature T for a total time of 1000 time steps. We can see that all of them exhibit a time drift. Figure 2 shows the time evolution of the same observables for a total time of 100000 time steps. We can see that after an initial time-drift they seem to fluctuate around constant values.

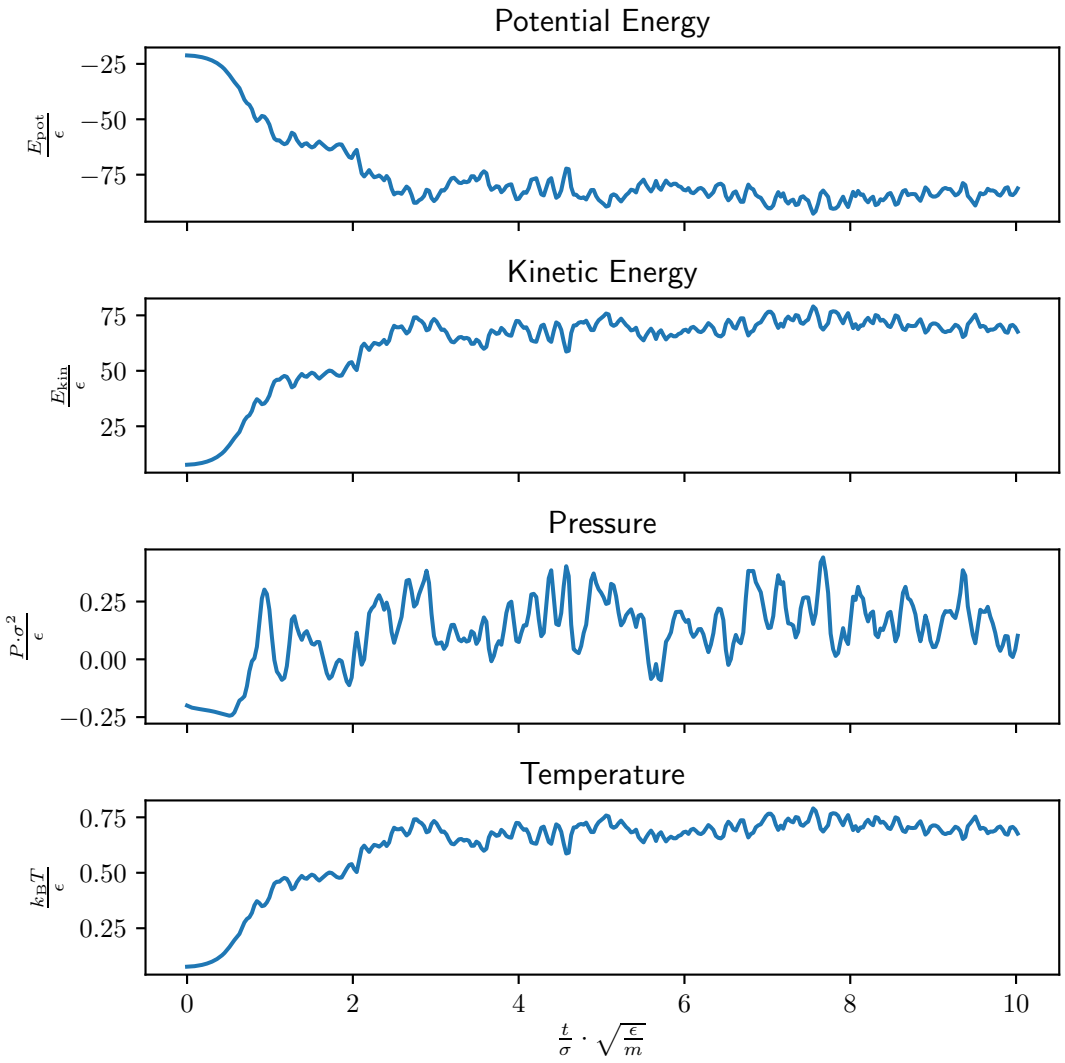

Figure 1: Time evolution of the potential energy E_{pot}, kinetic energy E_{kin}, pressure P and temperature T for a total time of 1000 time steps.

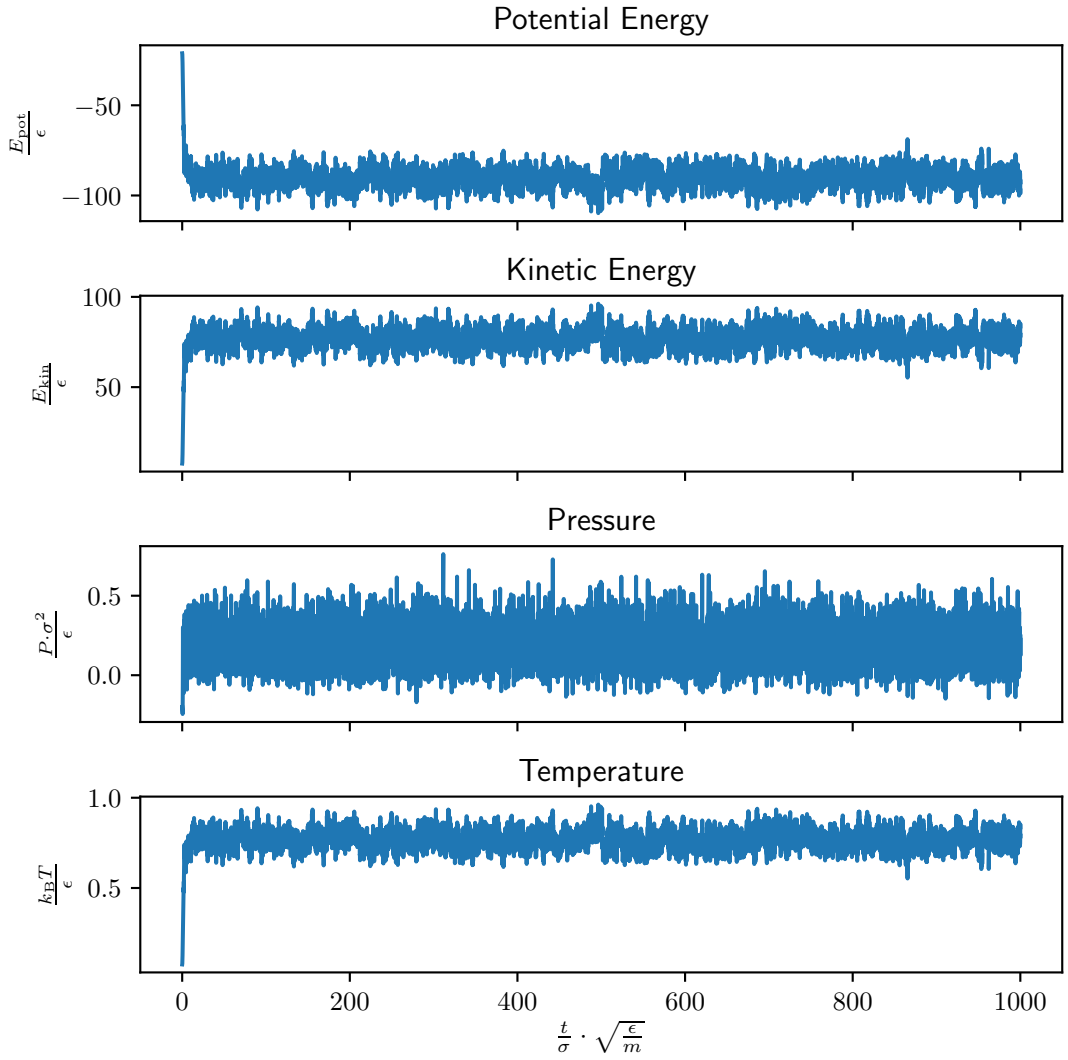

Figure 2: Time evolution of the potential energy E_{pot}, kinetic energy E_{kin}, pressure P and temperature T for a total time of 100000 time steps.

The function to calculate the running average of an observable is implemented in this fashion:

```
def running_average(O, M):
    ret = np.empty_like(O)
    N = len(O)
    for i in range(0, N):
        if (i < M) or (i >= N - M):
            ret[i] = np.nan
        else:
            ret[i] = np.sum(O[i-M:i+M+1]) / (2*M + 1)
    return ret
```

The function has as its arguments the time series of the observable O in the form of a NumPy array as well as the window size. It returns a NumPy array of the same size as the original time series, for all times for which the running average cannot be calculated, the value is set to np.nan. To sum all values in the window, the NumPy function np.sum is used.

Figure 3 shows the time evolution of the observables for a total time of 100000 time steps.

Furthermore, the running averages are shown for window sizes of $M = 10$ and $M = 100$. We can see that the running averages fluctuate less than the raw data and that the running average for $M = 100$ fluctuates less than the one for $M = 10$. The initial time drift of the observables corresponds to the equilibration of the system, it takes place on a time scale of about

$$t_{\text{eq}} \approx 10 \cdot \sigma \cdot \sqrt{\frac{m}{\epsilon}}.$$

After this equilibration time, the observables fluctuate around approximately constant values.

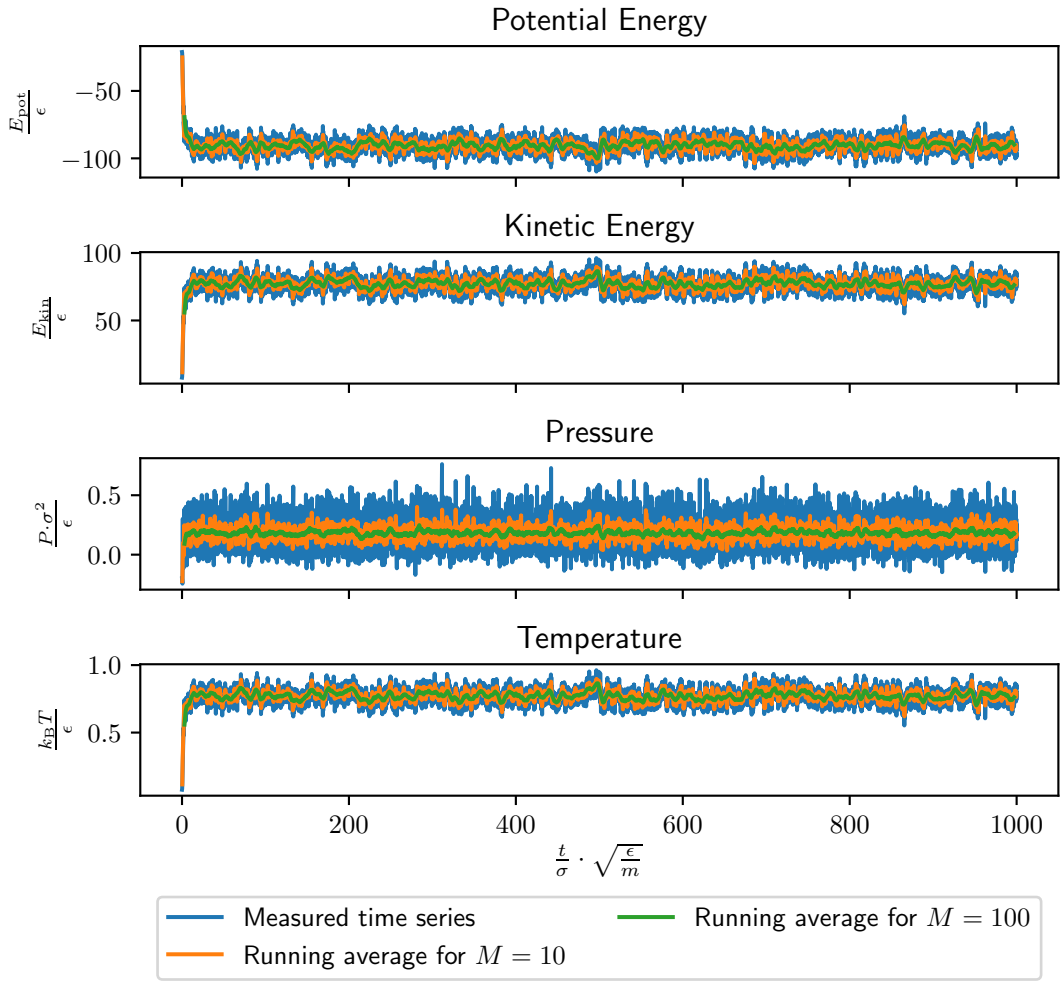

Figure 3: Time evolution of the potential energy E_{pot}, kinetic energy E_{kin}, pressure P and temperature T for a total time of 1000 time steps as well as the running averages for window sizes of $M = 10$ and $M = 100$.

To compute the mean values of the observables after an equilibration time t_{eq}, the function print_teq() was added:

```
def print_teq():
    if not hasattr(args, "t_eq"):
        return
```

```
    for o in observables:
        d = data[o]
        d = d[args.t_eq:]
        print("Average␣equilibrium␣%s:␣%s" % (o, sum(d) / len(d)))
```

The function contains a for-loop which runs over all observables and computes their equilibrium mean value (mean value from t_{eq} to the end of measurement) and outputs it. The measured equilibrium mean values are:

$$\begin{aligned}
\langle E_{\mathrm{pot}}/\epsilon \rangle &\approx -90.63 \\
\langle E_{\mathrm{kin}}/\epsilon \rangle &\approx 77.06 \\
\left\langle \frac{P\sigma^2}{\epsilon} \right\rangle &\approx 0.1831 \\
\left\langle \frac{k_{\mathrm{B}}T}{\epsilon} \right\rangle &\approx 0.7706.
\end{aligned}$$

4 Molecular Dynamics at a Desired Temperature

The temperature $T(t)$ of the system at a given time t is given by the equipartition theorem:

$$T(t) = \frac{1}{k_\mathrm{B} DN} \sum_{i=1}^{N} \frac{m}{2} \mathbf{v}^2. \tag{1}$$

To reach the desired temperature T_0, we introduce rescaled velocities:

$$\mathbf{v} \to a \cdot \mathbf{v} \tag{2}$$

with a scaling factor a. This results in

$$T_0 = \frac{1}{k_\mathrm{B} DN} \sum_{i=1}^{N} \frac{m}{2} (a \cdot \mathbf{v})^2 = a^2 \cdot \frac{1}{k_\mathrm{B} DN} \sum_{i=1}^{N} \frac{m}{2} \mathbf{v}^2 = a^2 \cdot T(t). \tag{3}$$

We can identify the scaling factor a as

$$a = \sqrt{\frac{T_0}{T(t)}}. \tag{4}$$

The rescaling function looks like this:

```
def rescale(self, T0):
        temp = self.temperature()
        factor = np.sqrt(T0 / temp)
        self.v *= factor
```

This function is then called every time the observables are measured. The velocity rescaling can be enabled with the command line option -t $[T_0]$.

The time evolution of the potential energy E_pot, kinetic energy E_kin, pressure P and temperature T for a total time of 210000 time steps as well as the running averages for window sizes of $M = 10$ and $M = 100$ are shown in Figure 4 – Figure 6. The simulations are first run for 10000 time steps with velocity rescaling and a target temperature of $T_0 = \{0.3, 1.0, 2.0\} \cdot \epsilon \cdot k_\mathrm{B}^{-1}$. Afterwards, the velocity rescaling is turned off and the time evolution is computed in the microcanonical ensemble for another 200000 time steps. In the beginning of the simulation, we observe rapid changes in the energy components and the temperature, these are cause by the velocity rescaling. We can see that the temperature and kinetic energy begin to fluctuate again when the velocity-rescaling-thermostat is turned off, this is expected since they are not rescaled every third step anymore. Furthermore we observe that the potential energy decreases during the time when the velocity rescaling is used. For a target temperature of $T_0 = 0.3 \cdot \epsilon \cdot k_\mathrm{B}^{-1}$, the following equilibrium mean values were measured:

$$\begin{aligned}
\langle E_\mathrm{pot}/\epsilon \rangle &\approx -221.4 \\
\langle E_\mathrm{kin}/\epsilon \rangle &\approx 30.60 \\
\left\langle \frac{P\sigma^2}{\epsilon} \right\rangle &\approx -0.029 \\
\left\langle \frac{k_\mathrm{B} T}{\epsilon} \right\rangle &\approx 0.306.
\end{aligned}$$

For a target temperature of $T_0 = 1.0 \cdot \epsilon \cdot k_\mathrm{B}^{-1}$, the following equilibrium mean values were measured:

$$\begin{aligned}
\langle E_\mathrm{pot}/\epsilon \rangle &\approx -81.32 \\
\langle E_\mathrm{kin}/\epsilon \rangle &\approx 102.0 \\
\left\langle \frac{P\sigma^2}{\epsilon} \right\rangle &\approx 0.313 \\
\left\langle \frac{k_\mathrm{B} T}{\epsilon} \right\rangle &\approx 1.020.
\end{aligned}$$

For a target temperature of $T_0 = 2.0 \cdot \epsilon \cdot k_\mathrm{B}^{-1}$, the following equilibrium mean values were measured:

$$\begin{aligned}
\langle E_\mathrm{pot}/\epsilon \rangle &\approx -64.81 \\
\langle E_\mathrm{kin}/\epsilon \rangle &\approx 203.6 \\
\left\langle \frac{P\sigma^2}{\epsilon} \right\rangle &\approx 0.839 \\
\left\langle \frac{k_\mathrm{B}T}{\epsilon} \right\rangle &\approx 2.036.
\end{aligned}$$

In all three cases the mean value of the measured temperature is close to the target temperature which was used for the velocity rescaling.

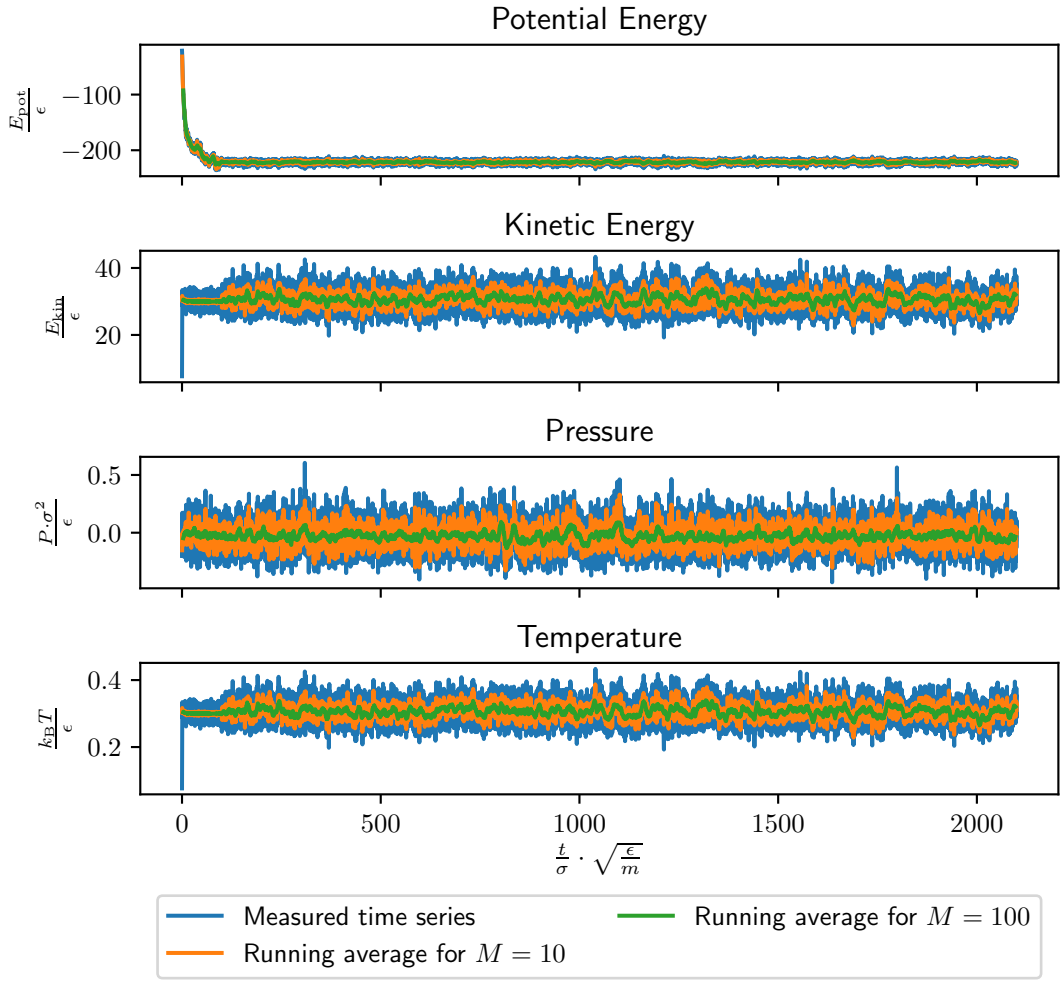

Figure 4: Time evolution of the potential energy E_{pot}, kinetic energy E_{kin}, pressure P and temperature T for a total time of 210000 time steps as well as the running averages for window sizes of $M = 10$ and $M = 100$. The simulation is first run for 10000 time steps with velocity rescaling and a target temperature of $T_0 = 0.3 \cdot \epsilon \cdot k_{\mathrm{B}}^{-1}$. Afterwards, the velocity rescaling is turned off and the time evolution is computed in the microcanonical ensemble for another 200000 time steps.

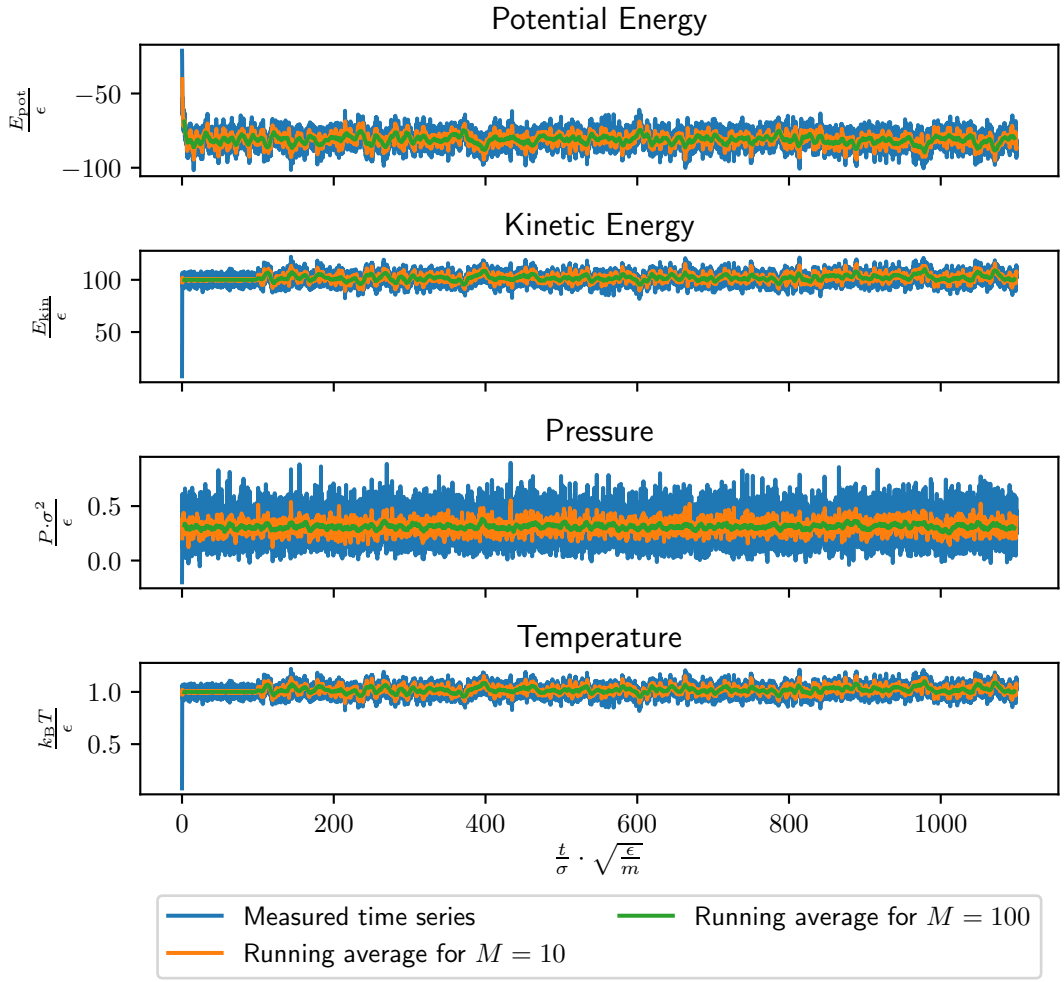

Figure 5: Time evolution of the potential energy E_{pot}, kinetic energy E_{kin}, pressure P and temperature T for a total time of 210000 time steps as well as the running averages for window sizes of $M = 10$ and $M = 100$. The simulation is first run for 10000 time steps with velocity rescaling and a target temperature of $T_0 = 1.0 \cdot \epsilon \cdot k_{\mathrm{B}}^{-1}$. Afterwards, the velocity rescaling is turned off and the time evolution is computed in the microcanonical ensemble for another 200000 time steps.

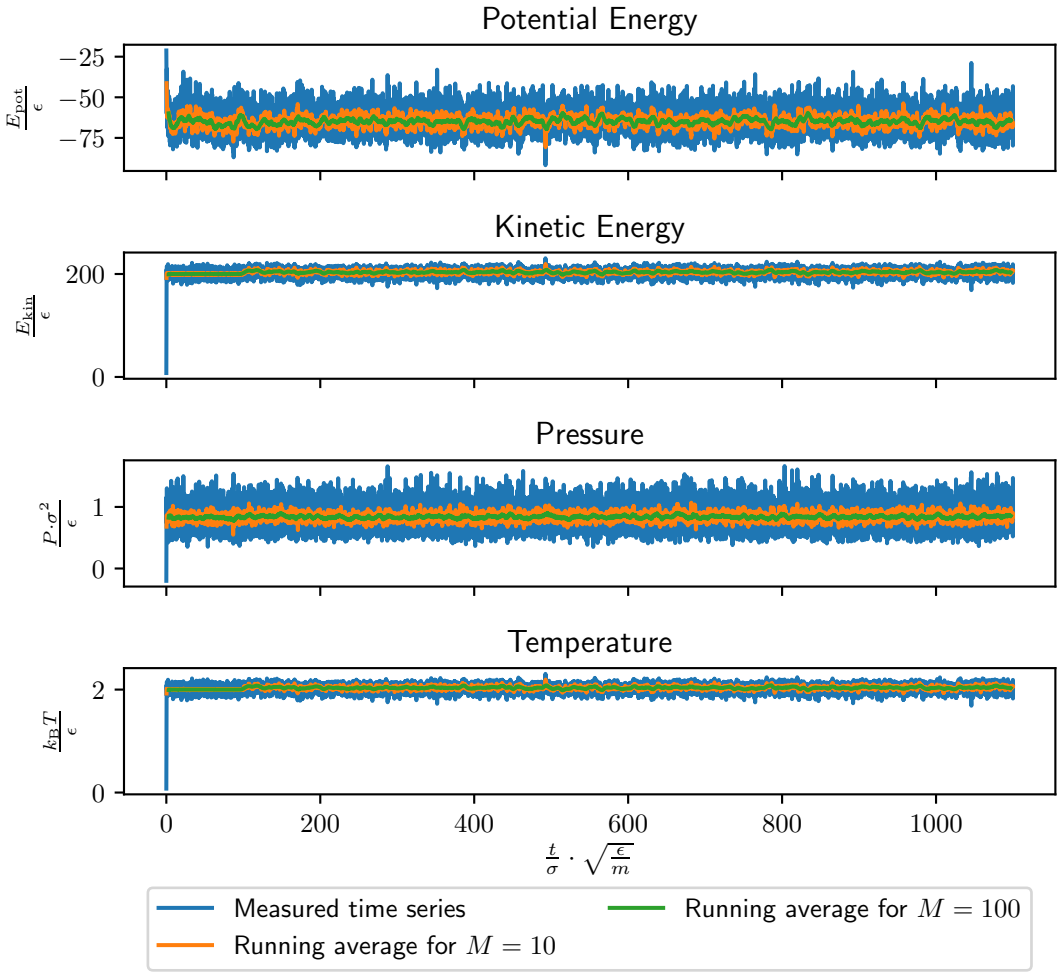

Figure 6: Time evolution of the potential energy E_{pot}, kinetic energy E_{kin}, pressure P and temperature T for a total time of 210000 time steps as well as the running averages for window sizes of $M = 10$ and $M = 100$. The simulation is first run for 10000 time steps with velocity rescaling and a target temperature of $T_0 = 2.0 \cdot \epsilon \cdot k_{\text{B}}^{-1}$. Afterwards, the velocity rescaling is turned off and the time evolution is computed in the microcanonical ensemble for another 200000 time steps.

5 Setting up and Warming up the System

In order to enable force capping, the function which computes the forces is modified in the following way:

```
def forces(self):
    # first update the distance vector matrix,
    # obeying minimum image convention
    self.distances()
    self.f_ij_matrix = self.r_ij_matrix.copy()
    r = np.linalg.norm(self.r_ij_matrix, axis=2)
    with np.errstate(all='ignore'):
        fac = np.where((r != 0.0) & (r < self.r_cut),\
        4.0 * (12.0 * np.power(r, -13.) - 6.0 * np.power(r, -7.)),\
        0.0)
    for dim in range(self.n_dims):
        with np.errstate(invalid='ignore'):
            self.f_ij_matrix[:, :, dim] *= np.where(r != 0.0,\
            fac / r,0.0)
    f = np.sum(self.f_ij_matrix, axis=0).transpose()
    self.f = np.clip(f, - self.f_max, self.f_max)
```

The NumPy function np.clip confines the force to the interval $[-f_{\max}, f_{\max}]$. When starting the simulation, an initial value for $f_{\max}$ can be set with the option $-f$. While the simulation is running, $f_{\max}$ is increase by 10% every time the observables are measured:

```
if args.fmax:
    sim.f_max *= 1.1
    if sim.f_max >= 1e9:
        sim.f_max = np.inf
```

Once a value of $f_{\max} = 10^9$ is reached, the force capping is turned off. To assign random initial particle positions, the following code is used.

```
x = np.random.random((DIM, N_PART))
for i in range(DIM):
    x[i] *= BOX[i]
```

The warming up with force-capping and velocity rescaling works as intended, details are discussed in section 7.

6 Radial Distribution Function

To calculate the radial distribution function at a given time, the following function was implemented:

```
def rdf(self):
    self.distances()
    r = np.linalg.norm(self.r_ij_matrix, axis=2)
    hist, bins = np.histogram(r, bins=100, range=(0.8, 5))
    return hist
```

First, the distances are calculated from the matrix of distance vectors. Next, a histogram is produced using the NumPy function np.histogram. Note that this histogram is not properly normalized. Because we are only interested in the average RDF, the normalization is performed in the same step as the averaging. The function to compute the average RDF beginning at a given time t_{eq} looks the following way:

```
def average_rdf(rdfs, t_eq):
    rdfs = rdfs[t_eq:]
    ret = np.zeros_like(rdfs[0])
    for i in range(len(rdfs)):
        ret += rdfs[i]
    ret = ret/len(rdfs)
    for i in range(0, 100):
        ret[i] /= 100*np.pi*DENSITY*(2*dr*(0.8+i*dr) + dr*dr)
    return ret
```

First, all of the histograms beginning with t_{eq} are added. Then, the normalization is performed: first, the obtained histogram is divided by the total number of added histograms. Next, another normalization is performed in order to obtain an actual RDF. The normalization can be derived the following way: The RDF measures, how many particles are in a radial bin $[r, r+\Delta r]$ compared to an ideal gas of the same average density. The number of particles in the bin $[r, r+\Delta r]$ for an ideal gas of density ρ is given by.

$$N\left([r, r+\Delta r]\right) = \rho \cdot \int_0^{2\pi} \mathrm{d}\phi \int_r^{r+\Delta r} \mathrm{d}r'\, r' = \pi\rho\left(2r\Delta r + \Delta r^2\right). \tag{5}$$

In the function average_rdf presented above there also appears an additional factor of 100, this number corresponds to the total number of particles and accounts for the fact that the function rdf computes the histogram over all distances (average over all particles).

7 Measuring Equilibrium Mean Values of the Observables

7.1 $T = 0.3 \cdot \epsilon \cdot k_\mathrm{B}^{-1}$

To warm up the system with random initial positions, a simulation with an initial force capping of $f_\mathrm{max} = 20.0 \cdot \epsilon \cdot \sigma^{-1}$ and velocity rescaling was run for 1000 time steps. A plot of the observables during this warming up is shown in Figure 7. We can see that both pressure and potential energy steeply decrease. After that, the force capping was removed and the system was simulated for another 5000 time steps with the velocity rescaling still activated. In the end, the system was simulated for another 100000 time steps with the velocity rescaling turned off. A plot of the different observables after the warming up is shown in Figure 8. We can see that the system is already equilibrated once the velocity rescaling is turned off.

The computed equilibrium mean values of the observables are:

$$\begin{aligned}
\langle E_\mathrm{pot}/\epsilon \rangle &\approx -222.5 \\
\langle E_\mathrm{kin}/\epsilon \rangle &\approx 30.54 \\
\left\langle \frac{P\sigma^2}{\epsilon} \right\rangle &\approx -0.02607 \\
\left\langle \frac{k_\mathrm{B}T}{\epsilon} \right\rangle &\approx 0.3054.
\end{aligned}$$

We can see that the mean temperature stays close to the target temperature.

The equilibrium mean value of the radial distribution function for this case is shown in Figure 9. The radial distribution function for this temperature resembles the radial distribution function of an (imperfect) crystal, it oscillates strongly even large values of the distance r. This result indicates that the Lennard-Jones-Fluid forms some kind of solid phase with long range-order for small temperatures.

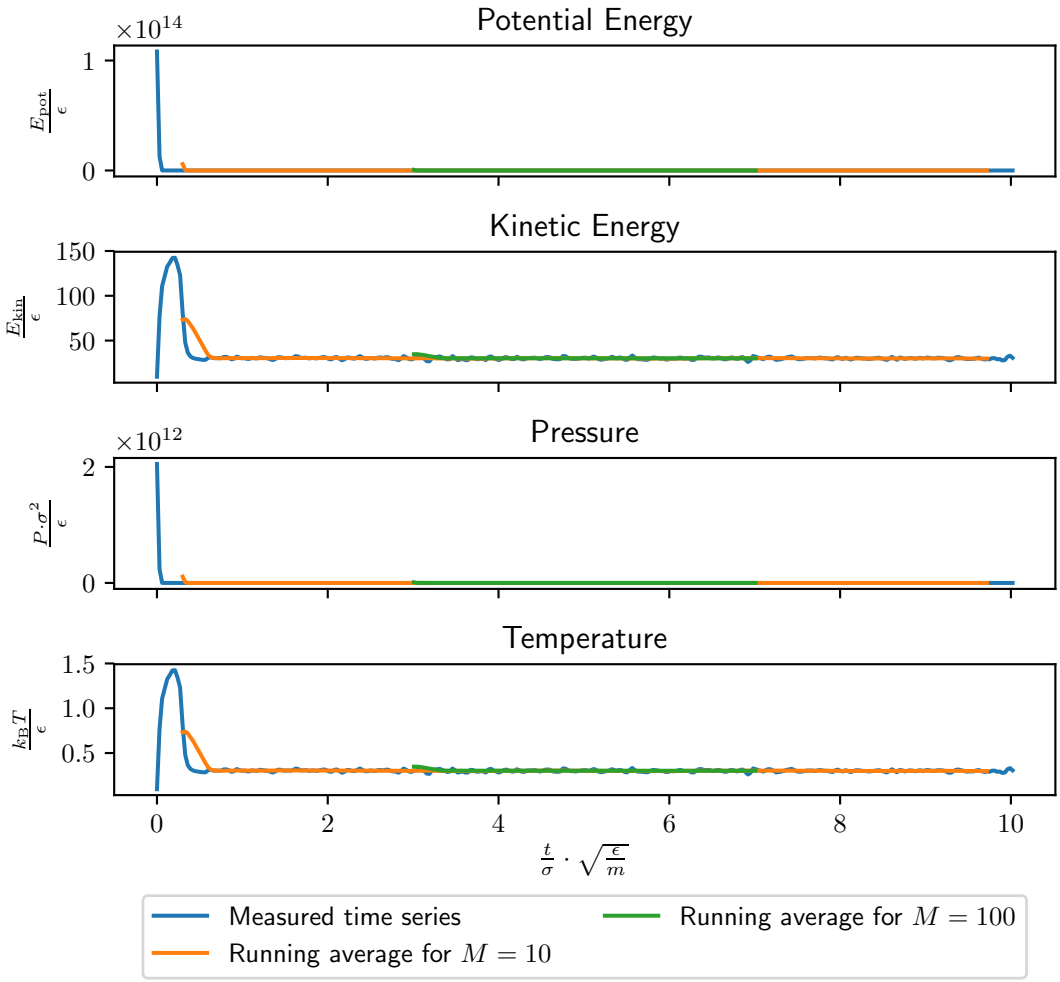

Figure 7: Time evolution of the observables during the warming up for the system with target temperature $T_0 = 0.3 \cdot \epsilon \cdot k_{\text{B}}^{-1}$.

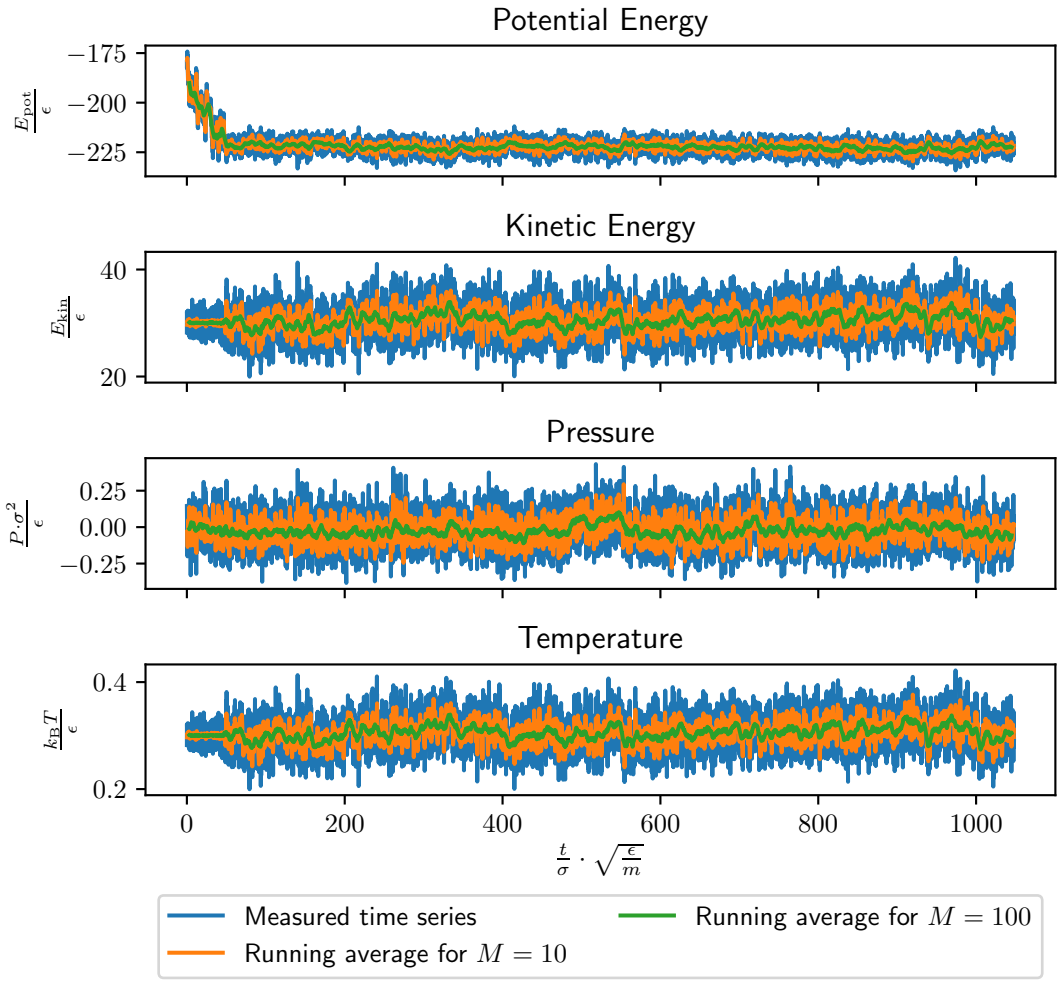

Figure 8: Time evolution of the observables during after warming up for the system with target temperature $T_0 = 0.3 \cdot \epsilon \cdot k_{\mathrm{B}}^{-1}$. During the 5000 time steps in the beginning, the velocity rescaling is turned on.

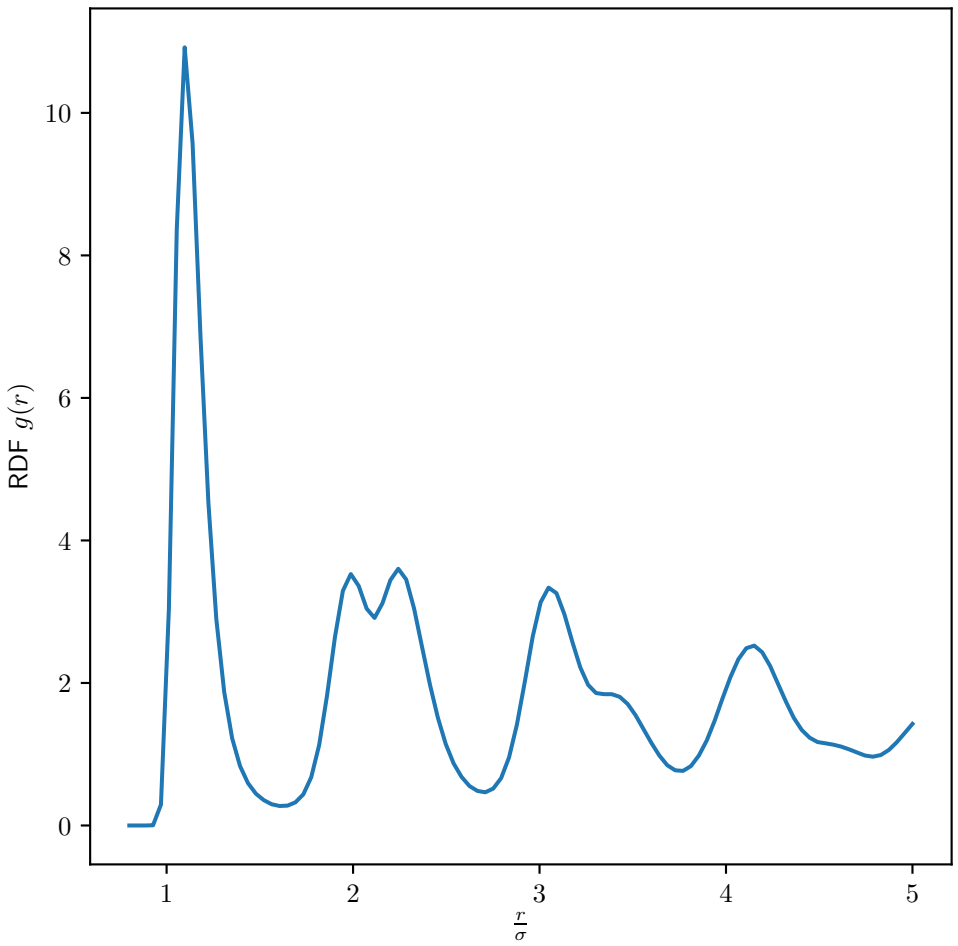

Figure 9: Equilibrium mean value of the radial distribution function for $k_{\mathrm{B}}T/\epsilon = 0.3$.

7.2 $T = 1.0 \cdot \epsilon \cdot k_{\mathrm{B}}^{-1}$

To warm up the system with random initial positions, a simulation with an initial force capping of $f_{\max} = 20.0 \cdot \epsilon \cdot \sigma^{-1}$ and velocity rescaling was run for 1000 time steps. A plot of the observables during this warming up is shown in Figure 10. We can see that both pressure and potential energy steeply decrease. After that, the force capping was removed and the system was simulated for another 6000 time steps with the velocity rescaling still activated. In the end, the system was simulated for another 100000 time steps with the velocity rescaling turned off. A plot of the different observables after the warming up is shown in Figure 11. We can see that the system is already equilibrated once the velocity rescaling is turned off.

The computed equilibrium mean values of the observables are:

$$\begin{aligned}
\langle E_{\mathrm{pot}}/\epsilon \rangle &\approx -79.07 \\
\langle E_{\mathrm{kin}}/\epsilon \rangle &\approx 105.2 \\
\left\langle \frac{P\sigma^2}{\epsilon} \right\rangle &\approx 0.3337 \\
\left\langle \frac{k_{\mathrm{B}}T}{\epsilon} \right\rangle &\approx 1.052.
\end{aligned}$$

We can see that the mean temperature stays close to the target temperature.

The equilibrium mean value of the radial distribution function for this case is shown in Figure 12. The radial distribution function for this temperature resembles the radial distribution function of a liquid, the oscillations quickly decay for large values of the distance r, the function approaches the value 1. This result indicates that the Lennard-Jones-Fluid forms a liquid phase with short range-order for this temperature.

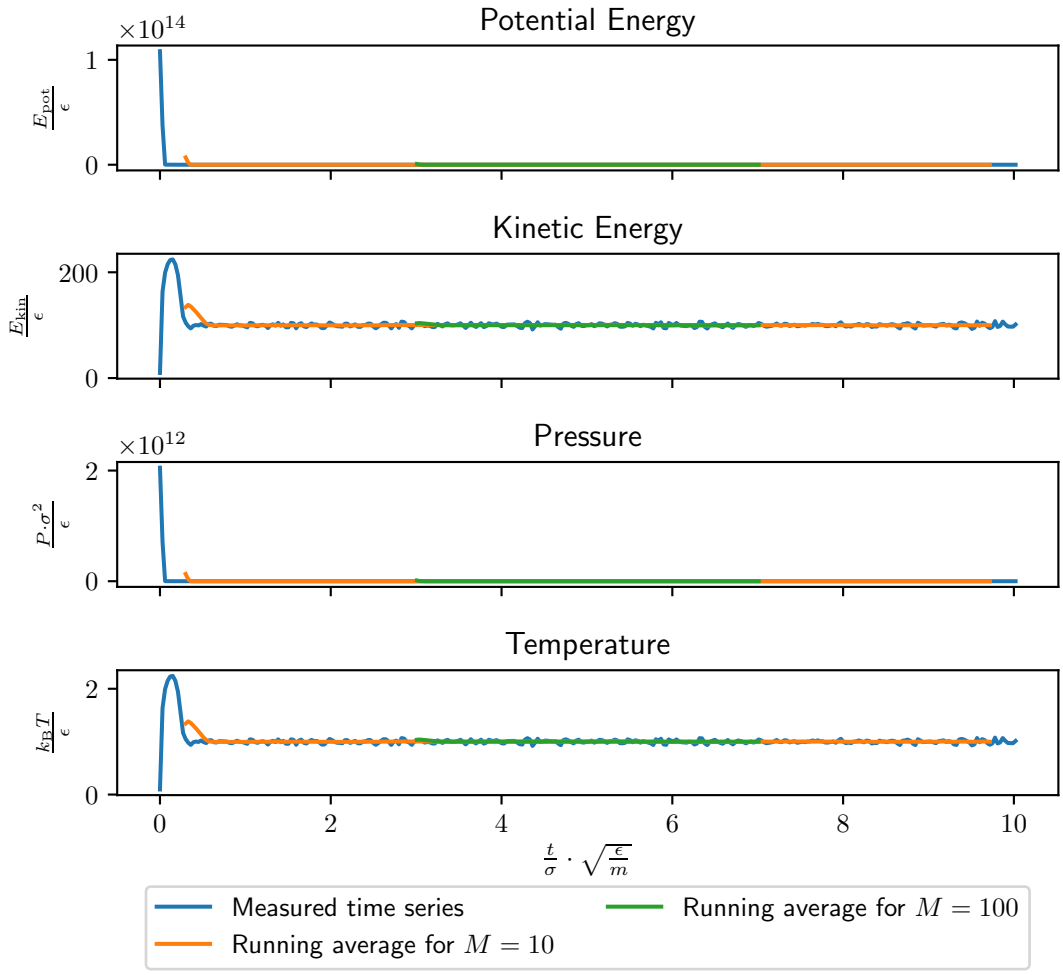

Figure 10: Time evolution of the observables during the warming up for the system with target temperature $T_0 = 1.0 \cdot \epsilon \cdot k_{\mathrm{B}}^{-1}$.

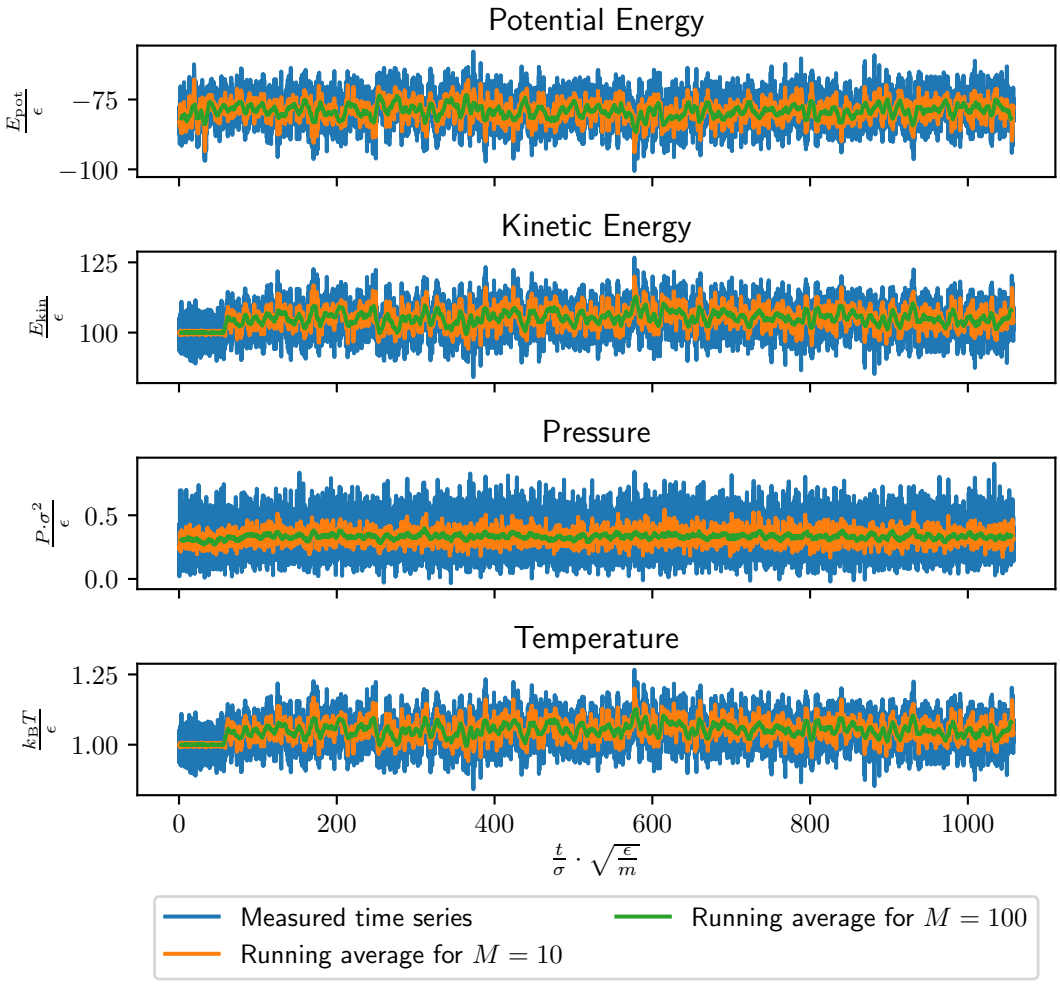

Figure 11: Time evolution of the observables during after warming up for the system with target temperature $T_0 = 1.0 \cdot \epsilon \cdot k_{\mathrm{B}}^{-1}$. During the 6000 time steps in the beginning, the velocity rescaling is turned on.

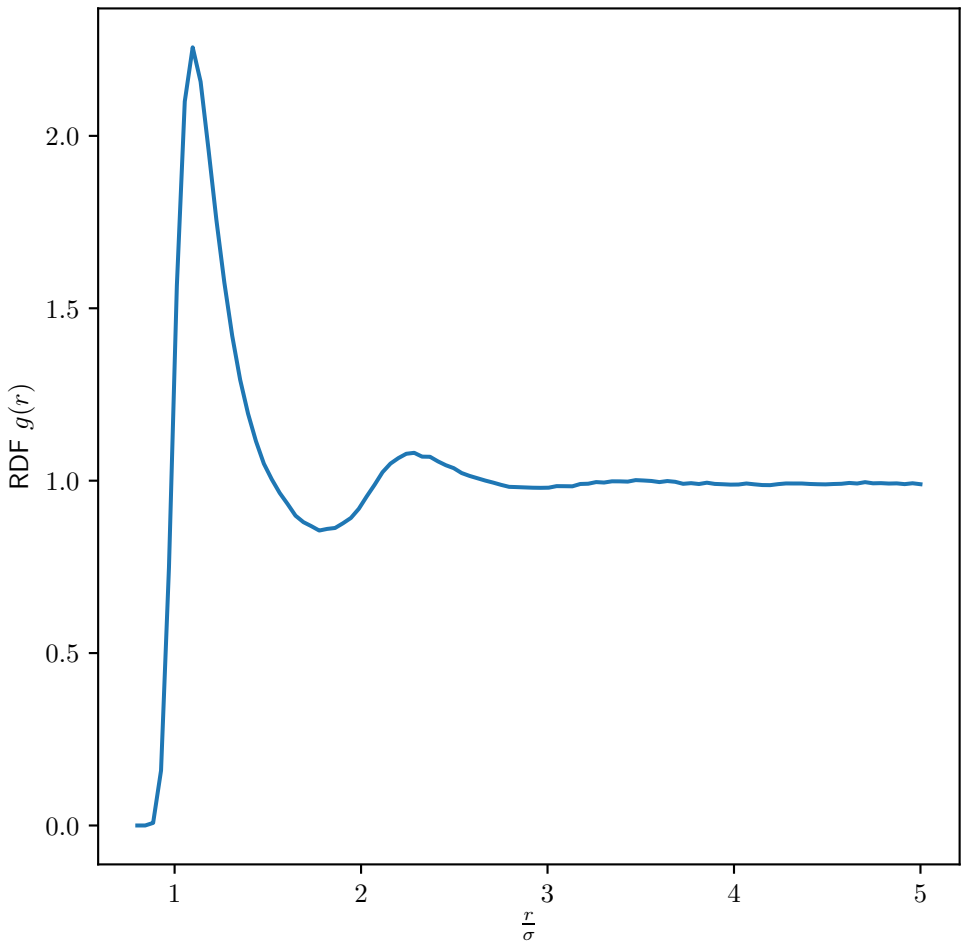

Figure 12: Equilibrium mean value of the radial distribution function for $k_{\mathrm{B}}T/\epsilon = 1.0$.

7.3 $T = 2.0 \cdot \epsilon \cdot k_{\mathrm{B}}^{-1}$

To warm up the system with random initial positions, a simulation with an initial force capping of $f_{\max} = 20.0 \cdot \epsilon \cdot \sigma^{-1}$ and velocity rescaling was run for 1000 time steps. A plot of the observables during this warming up is shown in Figure 13. We can see that both pressure and potential energy steeply decrease. After that, the force capping was removed and the system was simulated for another 6000 time steps with the velocity rescaling still activated. In the end, the system was simulated for another 100000 time steps with the velocity rescaling turned off. A plot of the different observables after the warming up is shown in Figure 14. We can see that the system is already equilibrated once the velocity rescaling is turned off.

The computed equilibrium mean values of the observables are:

$$\begin{aligned}
\langle E_{\mathrm{pot}}/\epsilon \rangle &\approx -63.80 \\
\langle E_{\mathrm{kin}}/\epsilon \rangle &\approx 206.8 \\
\left\langle \frac{P\sigma^2}{\epsilon} \right\rangle &\approx 0.8710 \\
\left\langle \frac{k_{\mathrm{B}}T}{\epsilon} \right\rangle &\approx 2.068.
\end{aligned}$$

We can see that the mean temperature stays close to the target temperature.

The equilibrium mean value of the radial distribution function for this case is shown in Figure 15. The radial distribution function for this temperature resembles the radial distribution function of a liquid, the oscillations quickly decay for large values of the distance r, the function approaches the value 1. The oscillations decay even faster than in the previous case, this indicates that the fluid approaches the characteristics of a gas as the temperature is increased.

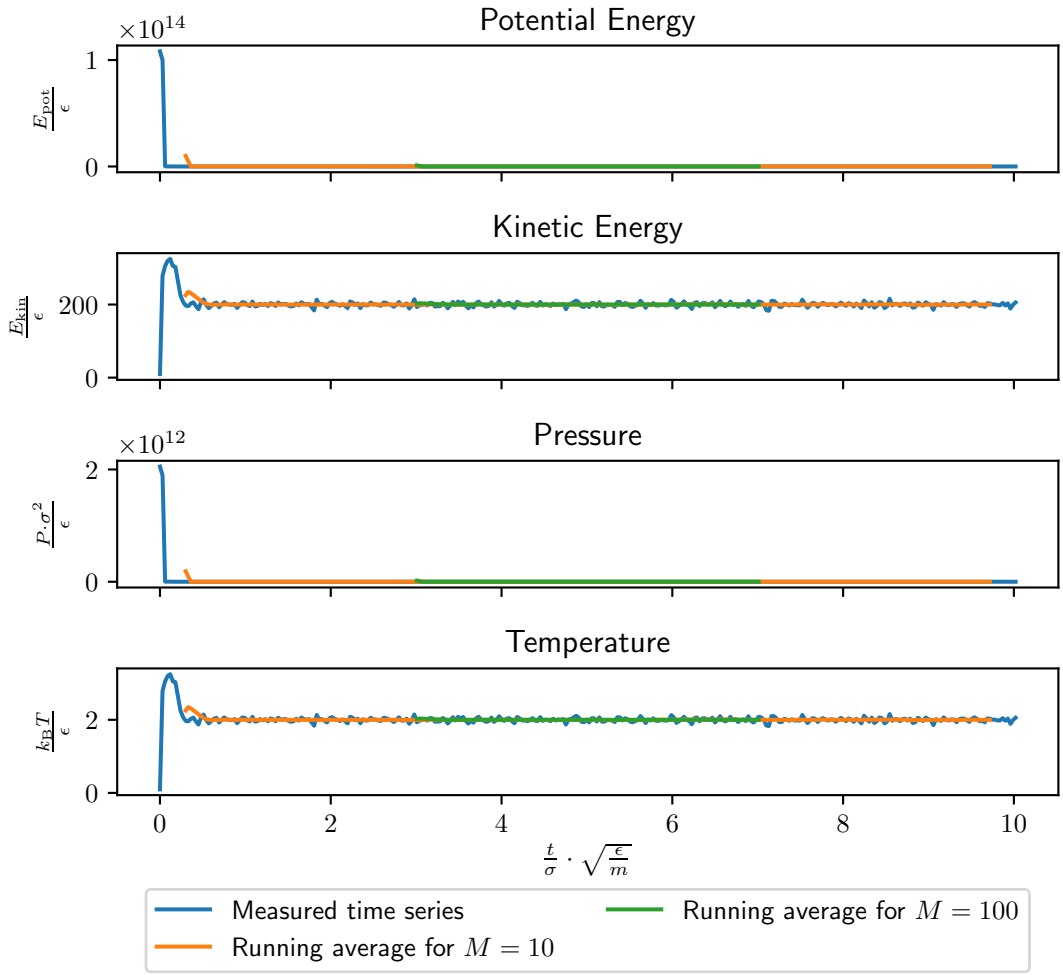

Figure 13: Time evolution of the observables during the warming up for the system with target temperature $T_0 = 2.0 \cdot \epsilon \cdot k_\mathrm{B}^{-1}$.

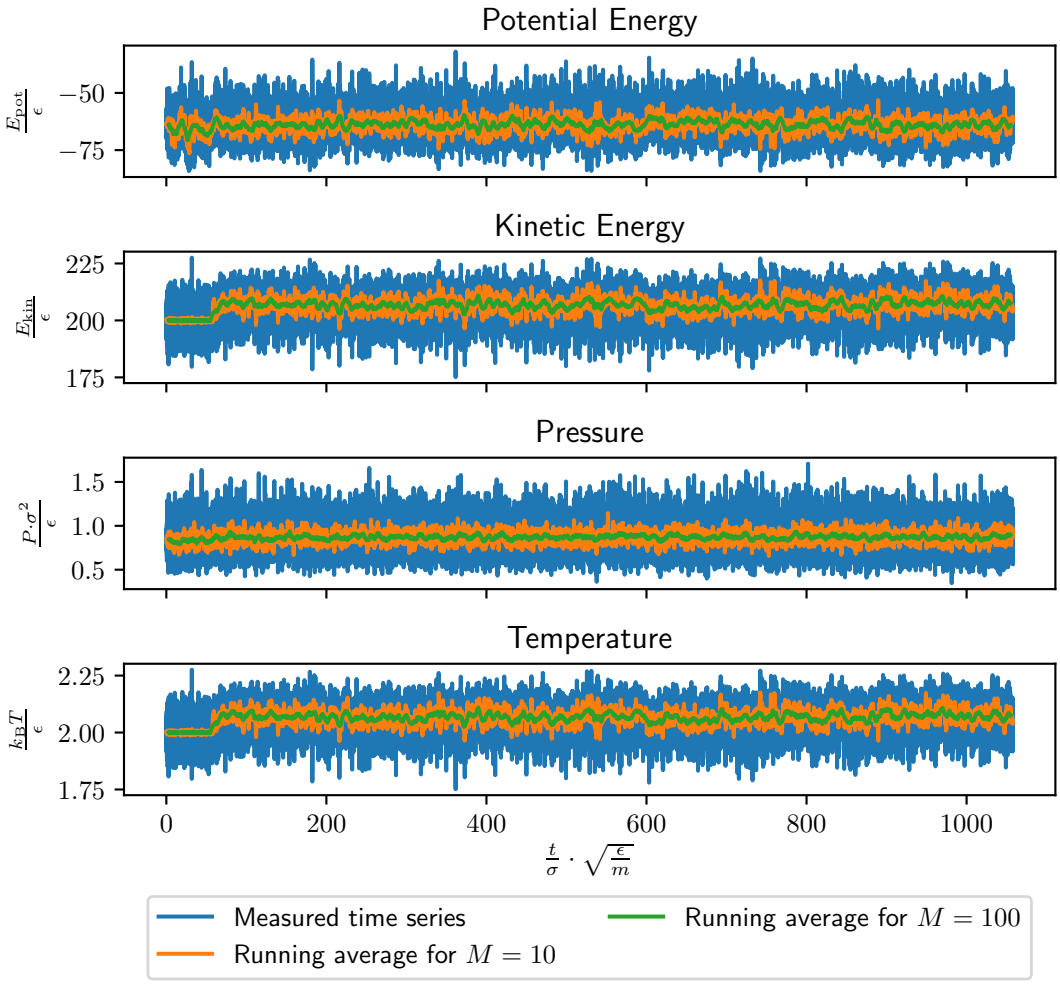

Figure 14: Time evolution of the observables during after warming up for the system with target temperature $T_0 = 2.0 \cdot \epsilon \cdot k_{\mathrm{B}}^{-1}$. During the 6000 time steps in the beginning, the velocity rescaling is turned on.

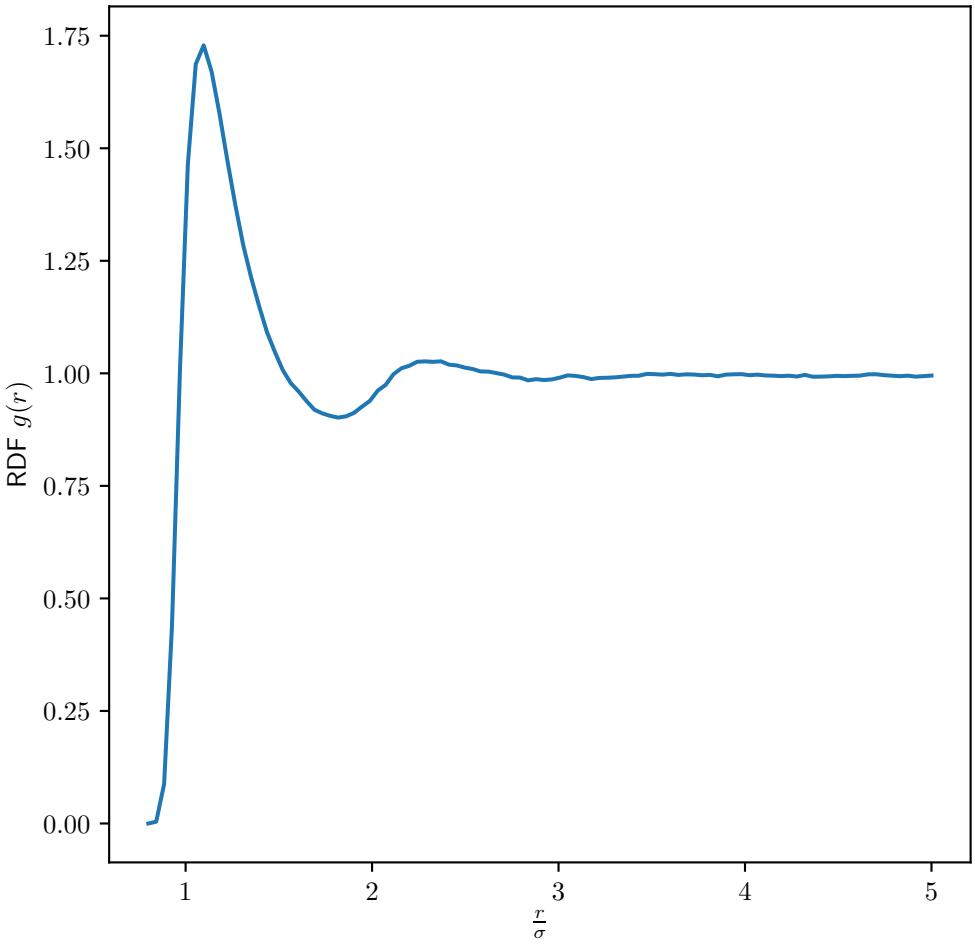

Figure 15: Equilibrium mean value of the radial distribution function for $k_{\mathrm{B}}T/\epsilon = 2.0$.

8 Tail Correction in the Pressure Calculation

The pressure for a system in d dimensions with volume V can be calculated using the expression

$$P = \underbrace{\frac{Nk_\mathrm{B}T}{V}}_{\text{ideal gas contribution}} + \underbrace{\frac{1}{dV}\sum_{i<j}\langle \mathbf{r}_{ij}\cdot\mathbf{F}_{ij}\rangle}_{\text{virial contribution}}. \tag{6}$$

We can split this expression into a part that corresponds to the system with truncated potentials and a correction ΔP:

$$P = \underbrace{\frac{Nk_\mathrm{B}T}{V} + \frac{1}{dV}\sum_{\substack{i<j\\ r_{ij}<r_\mathrm{c}}}\langle \mathbf{r}_{ij}\cdot\mathbf{F}_{ij}\rangle}_{P_\text{truncated}} + \underbrace{\frac{1}{dV}\sum_{\substack{i<j\\ r_{ij}>r_\mathrm{c}}}\langle \mathbf{r}_{ij}\cdot\mathbf{F}_{ij}\rangle}_{\Delta P}. \tag{7}$$

This leads to the expression

$$\Delta P = \frac{1}{3V}\sum_{\substack{i<j\\ r_{ij}>r_\mathrm{c}}}\langle \mathbf{r}_{ij}\cdot\mathbf{F}_{ij}\rangle \tag{8}$$

for a system in 3 dimensions. The average $\langle \mathbf{r}_{ij}\cdot\mathbf{F}_{ij}\rangle$ can in general be calculated using the appropriate statistical mechanical distribution function (for example the Boltzmann distribution for the canonical ensemble). Using the simplifying assumption that the particle density $\rho(\mathbf{r})$ is constant and given by

$$\rho(\mathbf{r}) = \frac{N}{V} \tag{9}$$

we can explicitly calculate $\langle \mathbf{r}_{ij}\cdot\mathbf{F}_{ij}\rangle$:

$$\begin{aligned}
\langle \mathbf{r}_{ij}\cdot\mathbf{F}_{ij}\rangle_{r_{ij}>r_\mathrm{c}} &\approx \frac{\rho}{N}\iiint\limits_{V\setminus B_{r_\mathrm{c}}(0)} \mathrm{d}^3 r_{ij}\,\mathbf{r}_{ij}\cdot\mathbf{F}_{ij} = \frac{1}{V}\iiint\limits_{V\setminus B_{r_\mathrm{c}}(0)} \mathrm{d}^3 r_{ij}\,24\epsilon\left(\frac{2\sigma^{12}}{r_{ij}^{12}} - \frac{\sigma^6}{r_{ij}^6}\right)\\
&\approx \frac{1}{V}\iiint\limits_{\mathbb{R}^3\setminus B_{r_\mathrm{c}}(0)} \mathrm{d}^3 r_{ij}\,24\epsilon\left(\frac{2\sigma^{12}}{r_{ij}^{12}} - \frac{\sigma^6}{r_{ij}^6}\right) = \frac{96\pi\epsilon}{V}\int_{r_\mathrm{c}}^{\infty}\mathrm{d}r_{ij}\,r_{ij}^2\left(\frac{2\sigma^{12}}{r_{ij}^{12}} - \frac{\sigma^6}{r_{ij}^6}\right)\\
&= \frac{96\pi\epsilon\sigma^3}{3V}\left(\frac{2}{3}\left(\frac{\sigma}{r_\mathrm{c}}\right)^9 - \left(\frac{\sigma}{r_\mathrm{c}}\right)^3\right)
\end{aligned} \tag{10}$$

In the third step we extended the range of integration over $\mathbb{R}^3 \setminus B_{r_\mathrm{c}}(0)$, this is approximately equal to the integral over $V \setminus B_{r_\mathrm{c}}(0)$ because the integrand goes to zero like r_{ij}^{-4} as $r_{ij} \to \infty$. Plugging this expression into the formula for the tail correction results in

$$\begin{aligned}
\Delta P &= \frac{1}{3V}\sum_{\substack{i<j\\ r_{ij}>r_\mathrm{c}}}\langle \mathbf{r}_{ij}\cdot\mathbf{F}_{ij}\rangle = \frac{1}{3V}\cdot\frac{N(N-1)}{2}\cdot\langle \mathbf{r}_{ij}\cdot\mathbf{F}_{ij}\rangle_{r_{ij}>r_\mathrm{c}}\\
&\approx \frac{1}{3V}\cdot\frac{N(N-1)}{2}\cdot\frac{96\pi\epsilon\sigma^3}{3V}\left(\frac{2}{3}\left(\frac{\sigma}{r_\mathrm{c}}\right)^9 - \left(\frac{\sigma}{r_\mathrm{c}}\right)^3\right)\\
&\approx \frac{16\pi\rho^2\epsilon\sigma^3}{3}\left(\frac{2}{3}\left(\frac{\sigma}{r_\mathrm{c}}\right)^9 - \left(\frac{\sigma}{r_\mathrm{c}}\right)^3\right).
\end{aligned} \tag{11}$$

In the third step we made use of the approximation $N(N-1) \approx N^2$ for large N.