# MOLECULAR DYNAMICS

**Computational Physics**
University of Groningen
Prof. dr. ir. P. R. Onck

**Robert Monden**
Student number: s3851117
Date: November 11, 2020

## 1 Introduction

Molecular Dynamics (MD) is a technique for studying the properties of solids, liquids, amorphous materials (e.g. metallic glasses, colloids and emulsions) and molecules using computer simulations. Unlike in thermodynamics, where the temperature $T$ is constant, $T$ is not constant, while the number of particles $N$, the volume $V$ and energy $E$ *are.*[1] A brief description on how MD works is given in Section 2.

The aim of this report is to explain how MD can be used to study physical processes, such as phase transitions. Specifically, we do this by analyzing the phase behavior of a Lennard-Jones fluid under temperature, pressure and density changes.

In order to do this, we must first implement thermostats in the given MD code. Then, we must calculate the pressure and analyze its effects over time when one of the two thermostats is activated. We then introduce the Berendsen barostat to the simulation. Finally, we change variables such as the temperature, pressure or density and analyze the effects. A more detailed description of the approach used is given in Section 2.

## 2 Method

This section explains what methodology was used in order to analyze the molecular dynamics simulation. Before the approach for analyzing the Lennard-Jones fluid is described, a brief introduction to MD is given.

### 2.1 Molecular Dynamics

This section discusses the essence of MD. The information described in this section borrows heavily from the lectures by professor Onck. See [1].

To calculate the acceleration of a particle, we need to look at the total force of all other particles. In MD we specifically look at two-body forces, which are forces generated by the interaction of two particles. These forces then cause particle $r_i$ to accelerate. This sum of all forces can be described mathematically using the following formula:

$$m\frac{d^2r_i}{dt^2} = F_i(r_0, \ldots, r_{N-1}) = \sum_{i<j=0}^{N-1} f_{ij},$$

where $i$ is subject to $0 \leq i \leq N-1$.

In other words, the net force on a particle $i$ due to the presence of a particle $j$ derives from a two-body potential $u(r_{ij})$ ($r_{ij}$ denotes the distance between two particles $i$ and $j$). To calculate this inter-particle force, we apply the following formula:

$$f_{ij} = -\frac{du(r_{ij})}{dr_{ij}}\left[\hat{e}_x\frac{x_i - x_j}{r_{ij}} + \hat{e}_y\frac{y_i - y_j}{r_{ij}} + \hat{e}_z\frac{z_i - z_j}{r_{ij}}\right]$$

The first term gives us the magnitude (slope) of the force, while the second term gives us the direction of this force. Note that $-\frac{du(r_{ij})}{dr_{ij}}$ is a conservative potential, which is the potential of a force that depends only on the starting and ending points of motion, not on the route taken.[2]

One fairly straightforward method for calculating the two-body potential $u(r_{ij})$ is the Leonard-Jones potential, which defines the average interaction between two atoms. It can be calculated as follows:

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

where $\sigma$ denotes the characteristic length scale in meters, $r$ the distance between the two particles and $\epsilon$ the strength in nanometers. $\epsilon$ and $\sigma$ can be deduced by fits to experimental data.[1]

The first term is repulsive, while the second term is attractive. As the two atoms move further apart, the attractive terms term becomes more important. This is due to Van der Waals intersection. The repulsive term is more important during the initial stages, when the electron clouds of the two atoms overlap. This in turn forces the atoms apart, due to Coulombic interaction. It follows that this is only dominates at very short distances.[1]

While the concept of MD is quite straightforward, it is computationally intensive. This is why usually only $10^6$ to $10^8$ particles are simulated. The disadvantage of this is that smaller numbers of particles are more likely to cause surface effects. The fewer particles there are, the more likely a particle is located near the surface. To combat these effects we can use PERIODIC BOUNDARY CONDITIONS. This involves limiting the simulation to one finite central box that is surrounded by an infinite number of replications of that box in all directions. Then, when a particle leaves the box, we bring back an image of that particle at the periodic boundary. By doing this we essentially mimic an infinite system using a finite, repeating sample.

When the distance between particles is large, the potential energy is low. They contribute very little to the total energy. We may therefore decide on a cut-off value, so we can ignore effects of these particles. For example, we may state that we only consider the energy of particles up to $r = 2\sigma$. When this distance is exceeded, we can say that $u(r) = 0$.

Since we are using Newton's second law, we need to integrate that in time. Due to the large number of particles and time steps, an efficient algorithm we could use is the VELOCITY-VERLET algorithm, which uses forward-difference approximation for both the position and velocity of particles simultaneously.

To calculate these for a particle $i$ at time $t + h$, we can use the following formulae:

$$\begin{aligned} r_i(t+h) &\approx r_i(t) + hv_i(t) + \frac{h^2}{2}F_i(t) \\ v_i(t+h) &\approx v_i(t) + h\overline{a(t)} \\ &\approx v_i(t) + h\left[\frac{F_i(t+h) + F_i(t)}{2}\right], \end{aligned}$$

where $\overline{a(t)}$ denotes the average acceleration, for which we used the average of forces at $t$ and $t + h$. The order is crucial here, since in order to calculate the forces for $v_i(t + h)$, we need the positions.

## 2.2 Program Stability

One of the first steps was analyzing the program's stability. This was done by first running the simulations for different values $h$: 0.0001, 0.001, 0.005, 0.01, 0.05 and 0.1 for values of $t$, subject to $0 \leq t \leq 10$. The maximum time was reduced from 100 to 10 in order to reduce the number of calculations needed, which would have taken several hours for $h = 0.0001$. The energy drift was then calculated for all these values of $h$.

Secondly, the simulation was run for various temperature values $t$: 5, 7, 9 and 11 with time step $h = 0.01$.

Finally, simulations were run for different values of the number of particles $N$: 50, 75, 100, 250.

In order to calculate and plot the energy drift for each time step $t$ during the simulations, the following formula was used:

$$\Delta E(n_{max}\Delta t) = \frac{1}{n_{max}} \sum_{n=1}^{n_{max}} \left| \frac{E(0) - E(n\Delta t)}{E(0)} \right|,$$

which can be expressed in pseudo code as:

```
nmax := length of total energy array
drift := []

for ts in length of time array:
    sum := 0

    for t in 0..ts:
        sum = sum + (E(0) - E(t)) / E(0)

    append (1/nmax) * sum to drift
```

During all three phases, the performance was recorded. This was done by calculating the number of time steps completed per second: $p = \frac{T_s}{T_r}$, where $T_s$ denotes the number of time steps and $T_r$ the number of seconds passed in real time.

## 2.3 Implementation of Thermostats

In the third, fourth and second assignments, thermostats were included within simulations. The two thermostats implemented were the ANDERSEN and BERENDSEN thermostats. Due to a mistake their implementation was included in the example code, therefore they will not be covered in this section.

### 2.3.1 Coupling Strength

To determine at what strengths the thermostats are weakly and strongly coupled, the ratios $\nu\Delta t$ and $\frac{\Delta t}{\tau_T}$ were used, respectively, for the Andersen and Berendsen thermostats.

#### 2.3.2 Mean Squared Displacement

To calculate the Mean Squared Displacement (MSD), the following formula was used:

$$MSD = \frac{1}{N}\sum_{i=1}^{N}\left|x^{(i)}(t) - x^{(i)}(0)\right|^2,$$

where $N$ denotes the number of particles and $x^{(i)}(t)$ the position of particle $i$ at time $t$. The pseudo code for this formula is as follows:

```
for particle in particles:
    dist_x := (current x - initial x)^2
    dist_y := (current y - initial z)^2
    dist_z := (current z - initial z)^2

    msd := sum(dist_x + dist_y + dist_z)/n
```

The MSD was then plotted as a function of time for various coupling strengths. This was done for both thermostats.

In order to obtain the diffusion coefficient $D$ in $MSD = 2nDt$, simple algebra was used, since variables $MSD$, $n$ and $t$ had known values. For example, let $MSD = 62.4$, $n = 3$, $t = 2$, then $D = 5.2$.

#### 2.3.3 Velocity Distributions

Velocity distributions were plotted for both thermostats, using various strengths. These were then compared to a Maxwell-Boltzmann distribution.

### 2.4 Phrase Transitions

In this phase of the assignment, pressure was introduced. In order to calculate the pressure $P$, the following formula was used:

$$P = \frac{\rho}{3N}(2E_k + w)$$

where $\rho$ denotes the density, $N$ the number of particles, $E_k$ the kinetic energy and $w$ the weighted average of the forces.

The formula was implemented as follows (for the actual Python code, see Appendix A):

```
w := sum(forces) * h
rho := n * box_size^3
pressure := (rho / (3 * n)) * (2 * KE + w)

pressure_values.append(pressure)
```

A graph of $P$ vs $t$ was then generated, displaying the pressure over time in a simulation where the Andersen thermostat was turned on half-way.

Then, in order to perform simulations for different particle densities, ten values of $\rho$ were selected. These values were evenly distributed between 0 and 1.5. Applying some elementary algebra resulted in the corresponding $N$ values that were converted to integers afterwards: $\rho = \frac{N}{s^3} \implies s^3\rho = N$. Standard settings for the simulation length, box size and $h$ were used.

### 2.5 Berendsen Barostat

The Berendsen barostat was implemented in a similar way to the Andersen and Berendsen thermostats. The Berendsen barostat would activate half-way and scale the positions. The following formula was used for scaling factor $\mu$:

$$\mu = \sqrt[3]{(1 + \frac{\Delta t}{\tau_p}(P - P_0))}$$

To implement this formula and have the barostat enable half-way, the following algorithm was used:

```
increase iterations

if iterations > activation time:
    mu := cubic_root(1+((h/strength) * (
        current_pressure - target_pressure
    )))

    return mu * positions

return positions
```

Simulations were run with a starting temperature of 5, box size of 15, $N = 50$, $h = 0.01$ and target pressure of 1.0. Although due to timing constraints this was not possible, a phase diagram would then be generated, which would also display the isotherms. A properly-functioning barostat would affect the density of the box.

## 3 Program Structure

While working on the final assignment, several adjustments and additions were made to the code. In this section these adjustments and additions are defended.

### 3.1 Thermostats

The example coded included thermostat code. Using the `thermostat()` function, specifying the name of a thermostat would apply this thermostat to the velocities and return these updated velocities. The function check the thermostat name and follow the code branch appropriate for that thermostat name.

However, if new thermostats are required, this method needs to be updated as well. To get around this problem, the COMMAND PATTERN was used to avoid the need for if-statements. Instead, any object that is an instance of the `Thermostat` class' sub classes, could be passed. This is because any class extending `Thermostat` is required to implement `get_velocities()`.

In later questions of the assignment simulations needed to be run with a thermostat being activated half-way. To accomplish this, a variation of the DECORATOR PATTERN was used. The `DormentThermostat` can automatically activate a given thermostat at a given time. Instances of this class can be passed as a thermostat argument. Whenever a new DormentThermostat is created, an activation timestep and Andersen or Berendsen thermostat are passed as arguments to the constructor. The DormentThermostat class keeps track of the number of times `get_velocities()` was invoked (this equals the number of iterations) and returns the unaltered velocities until the activation timestep is reached. After this point, the thermostat that was specified in the beginning is applied to the velocities.

### 3.2 Simulation Options

In order to properly study the effects of manipulating different variables within the ensemble, a large number of simulations needed to be run. These simulations in turn required a large number of parameters. Most of the time these parameters would remain unchanged. Therefore, in order to reduce boilerplate code, a class was introduced containing simulation settings. Functions requiring eight different parameters would henceforth require only one parameter. This class — the `SimulationOptions` class — also contains default options.

### 3.3 Multiprocessing

In multiple places, multiprocessing was used to speed up computations. For example, in order to test thermostat strengths, four different thermostat strengths were used. These simulations did not depend on each other, hence they could be run in a parallel manner. Since Python does not support threading, the `multiprocessing` package was used. This package enables parallelism through multiprocessing.[3]

### 3.4 Simulation Stability

Program stability was improved using several measures. First, in the `force_function` function, the line `if rd == 0.` was replaced by `if rd < 0.001` after simulations for assignment 1 were run.

## 4 Results and Discussion

This section displays the results of the simulations without further discussing them. The answers to the questions in the document are also answered here.

**Note:** All questions are answered in the order they appear. The corresponding question is stated within the parentheses following the answer.

### 4.1 Program Stability

Figures 1 to 18 in the appendix display what happens when $h$ is set to 0.0001, 0.001, 0.005, 0.01, 0.05, 0.1. Although in real life energy is always conserved, this is not the case in numerical simulations. This is due to round-off errors. The energy drift indicates the loss (or addition) of energy. As visualized, energy drift start at 0 and increases over time. This makes sense, as round-off errors keep building up as more and more iterations are run. (2.1)

Due to the stability of the simulation, energy (drift) and temperature may vary greatly between simulations. This is visible in Figure 9, where the total energy equals approximately $6 \cdot 10^2 1$. The energy drift is also much greater. (2.1)

Figures 31 and 32 display, respectively, the effects of increasing $N$ on the total and potential energy. As can be seen, the amount of energy is higher for higher values of $N$, whereas it is lower for lower values of $N$. This makes sense, since the potential energy is calculated using by inter-particle forces $r_{ij} \cdot f_{ij}$. It therefore stands that the more particles, the greater the force, which means greater potential energy. (2.3b) For each value of $N$ the performance was recorded:

- $N = 50$: 46 time steps per second;
- $N = 75$: 17 time steps per second;
- $N = 100$: 9 time steps per second;
- $N = 125$: 6 time steps per second;
- $N = 250$: 1 time step per second (graph not included).

While periodic boundary conditions mitigate the performance issues with large particle numbers somewhat, the time it takes to run a single time step increases quite rapidly. (2.3a)

## 4.2 Thermostats

This section covers the results of assignment 3.

### 4.2.1 Coupling Strength

For the Andersen thermostat, the value of $\nu\Delta t$ increases as $\nu$ (the thermostat strength) increases. The opposite is true for the Berendsen thermostat, where the value of $\frac{\Delta t}{\tau_T}$ increases as $\tau_T$ approaches zero. (3.1)

It can therefore be said that the Andersen thermostat is weakly coupled at thermostat strengths closer to zero, whereas it is strongly coupled at higher values. The Berendsen thermostat is strongly coupled at thermostat strengths close to zero, whereas it is weakly coupled at higher thermostat values. (3.1)

### 4.2.2 Mean Squared Displacement

The figures in Appendix B.2.1 show the MSD value over time while the thermostats are active. At a thermostat strength of 5000, it can be seen that the MSD value fluctuates more heavily than the Andersen thermostat. This is due to the coupling strength of the Andersen thermostat, which is high at high thermostat strengths. The opposite can be seen in Figures 3 37 and 38, where high MSD values can be seen for the Andersen thermostat. (3.6c; 3.6f)

In theory, the MSD should plateau beyond a certain time. This indicates that particles move less quickly, possibly due to weaker inter-particle forces. (3.6d)

### 4.2.3 Velocity Distributions

Appendix B.2.2 and B.2.3 show the velocity distributions for, respectively, the Andersen and Berendsen thermostats. It can be seen that, at lower values of the thermostat strength, the normal distribution fits the velocity distribution more closely. At higher thermostat strengths, this is not the case, there the frequency of a velocity greatly exceeds the frequency predicted by the Maxwell-Boltzmann distribution. It can therefore be said that the velocity distributions are affected by the coupling strength. (3.7c; 3.7d)

## 4.3 Phase Transitions

Figure 51 in the appendix shows the pressure over time in a system where the Andersen thermostat is turned on half-way. As can be seen, enabling the Berendsen thermostat causes the pressure to decrease rapidly, after which it only fluctuates slightly. With the Andersen thermostat it takes more time for the pressure to decrease. Pressure fluctuations are also higher when the Andersen thermostat is enabled. (4.1.c; 4.3)

## 4.4 Berendsen Barostat

Figure 52 in the appendix displays the pressure over time when the Berendsen barostat is enabled halfway. Unfortunately, the barostat implementation did not work properly, hence it shows no effect on the pressure.

# 5 Conclusion

Having analyzed the effects on a Lennard-Jones fluid during temperature, pressure and density changes, we can now look back at the goal that was defined in the introduction: explaining how MD can be used to study physical processes.

Through thermostats and barostats we can measure the effects of changing temperatures and pressure. We can study the effects of changing the temperature to 0 degrees Celsius or to 2000 degrees. Similarly, we can simulate the effects of high pressure or low pressure on the energy levels. These are just a few examples of how MD can help us study physical processes.

# 6 References


[1] Patrick R. Onck. *Computational Physics: Lecture 3*. Lecture slides. 2020.

[2] Lumen. *Conservative Forces and Potential Energy*. N.d. URL: https://courses.lumenlearning.com/physics/chapter/7-4-conservative-forces-and-potential-energy/.

[3] Melvin Koh. *Concurrent Programming in Python is not what you think it is.* 2018. URL: https://hackernoon.com/concurrent-programming-in-python-is-not-what-you-think-it-is-b6439c3f3e6a.

# A Implementation of Pressure Formula

```
if kin_energy is not None:
    w: float = np.sum(rf)
    density: float = num_particles / box_size**3
    pressure = (density / (3 * num_particles)) * (2 * kin_energy + w)
    pressure_values[i] = pressure

```

# B Graphs

## B.1 Assignment 2

### B.1.1 H=0.0001

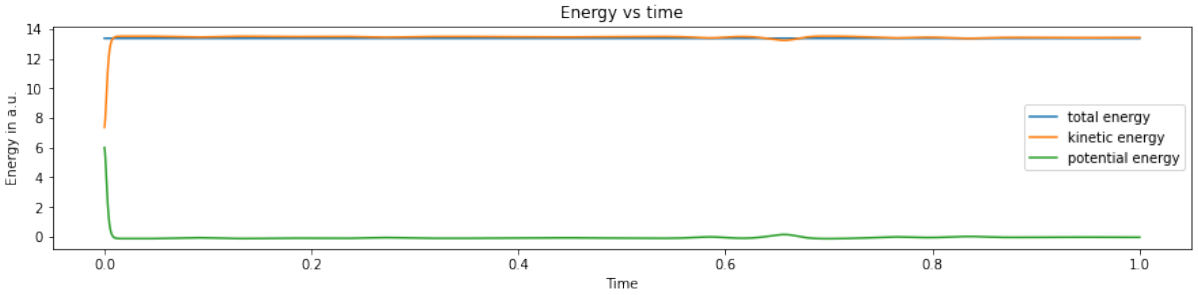


Figure 1: Total, potential and kinetic energy vs time for $h = 0.0001$.

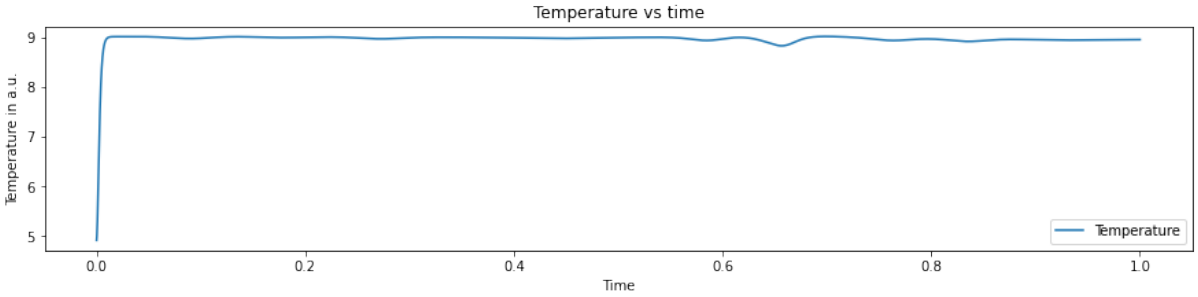


Figure 2: Temperature vs time for $h = 0.0001$.

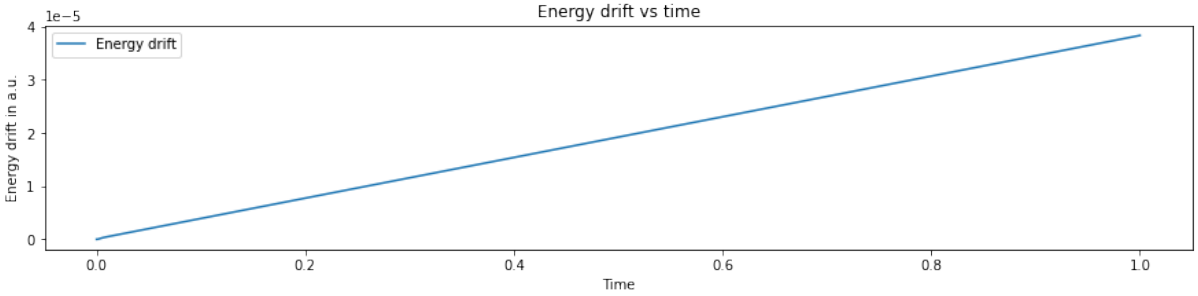


Figure 3: Energy drift for $h = 0.0001$.

### B.1.2 H=0.001

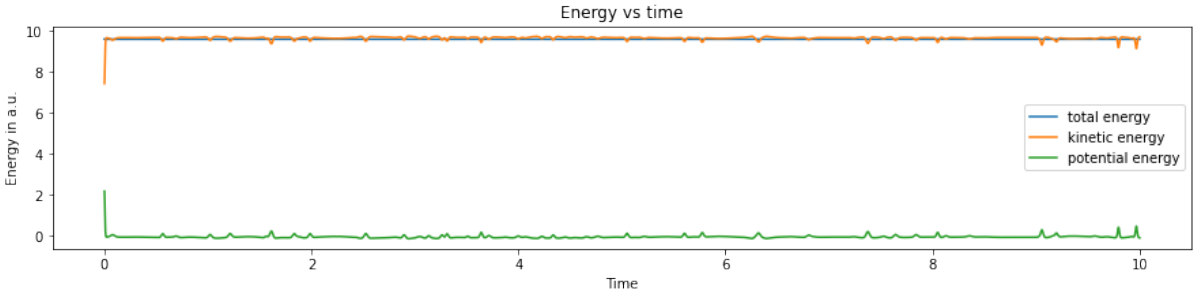


Figure 4: Total, potential and kinetic energy vs time for $h = 0.001$.

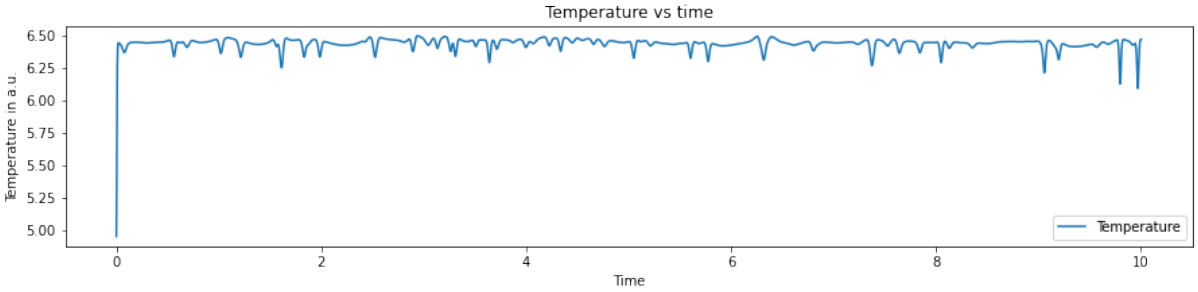


Figure 5: Temperature vs time for $h = 0.001$.

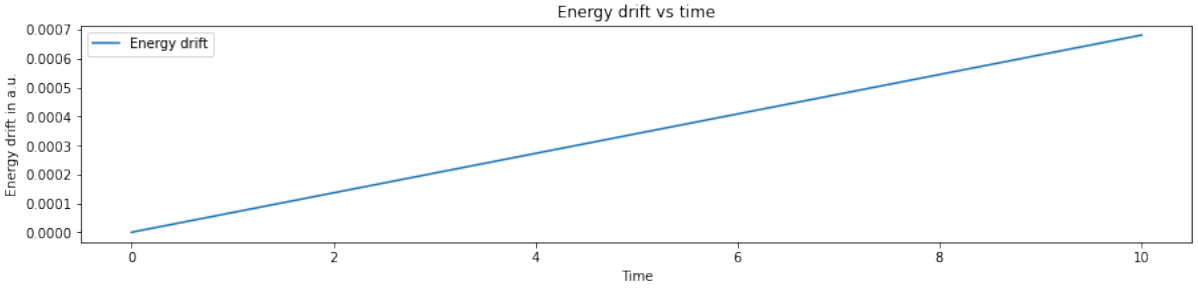


Figure 6: Energy drift for $h = 0.001$.

### B.1.3 H=0.005

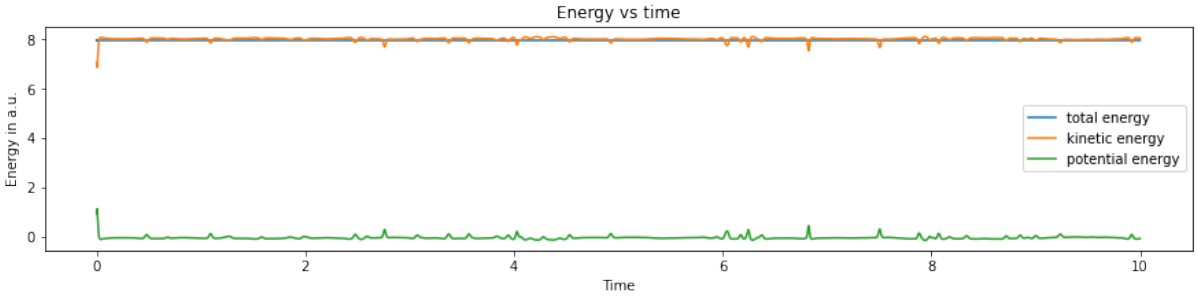


Figure 7: Total, potential and kinetic energy vs time for $h = 0.005$.

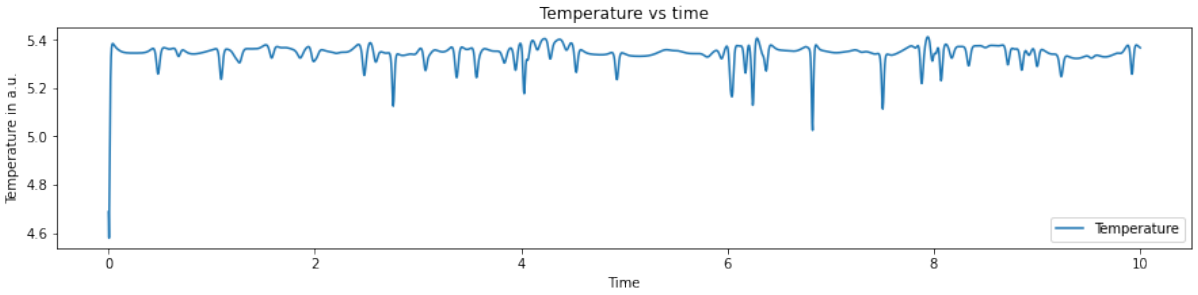


Figure 8: Temperature vs time for $h = 0.005$.

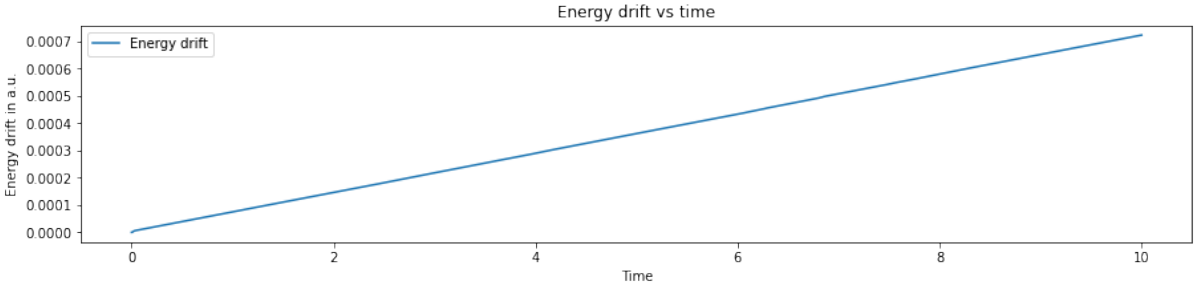


Figure 9: Energy drift for $h = 0.005$.

### B.1.4 H=0.01

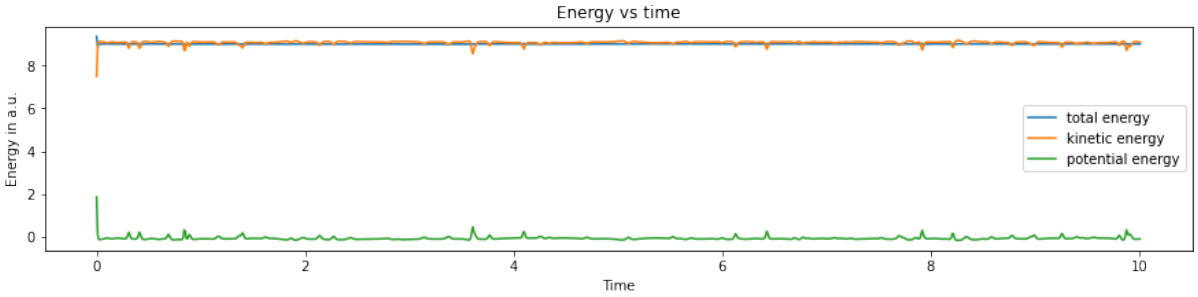


Figure 10: Total, potential and kinetic energy vs time for $h = 0.01$.

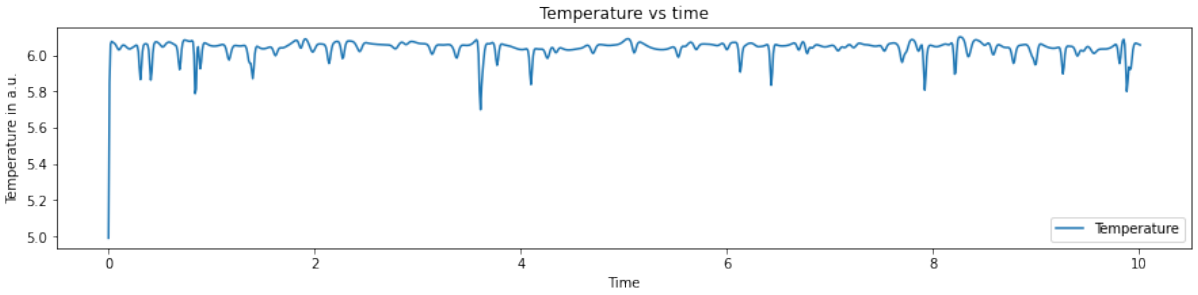


Figure 11: Temperature vs time for $h = 0.01$.

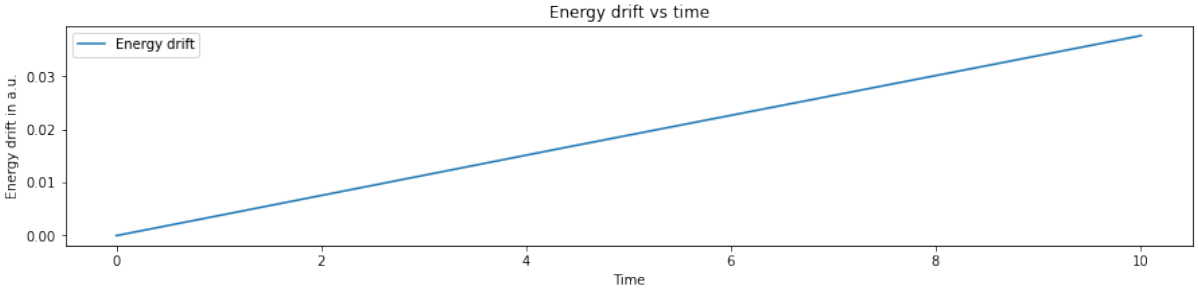


Figure 12: Energy drift for $h = 0.01$.

### B.1.5 H=0.05

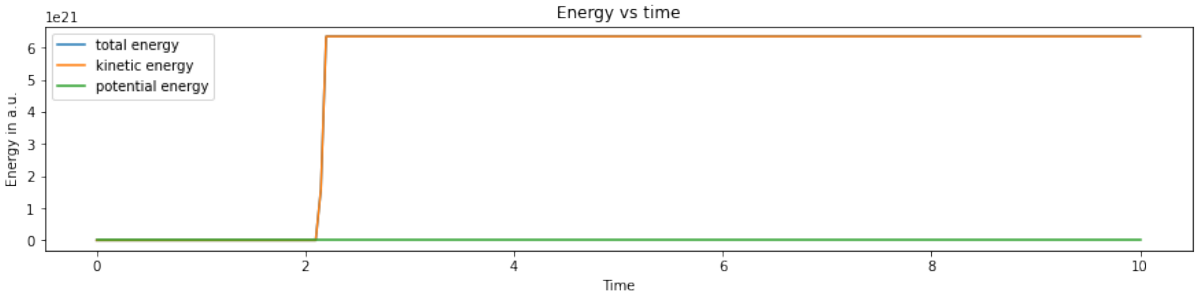


Figure 13: Total, potential and kinetic energy vs time for $h = 0.05$.

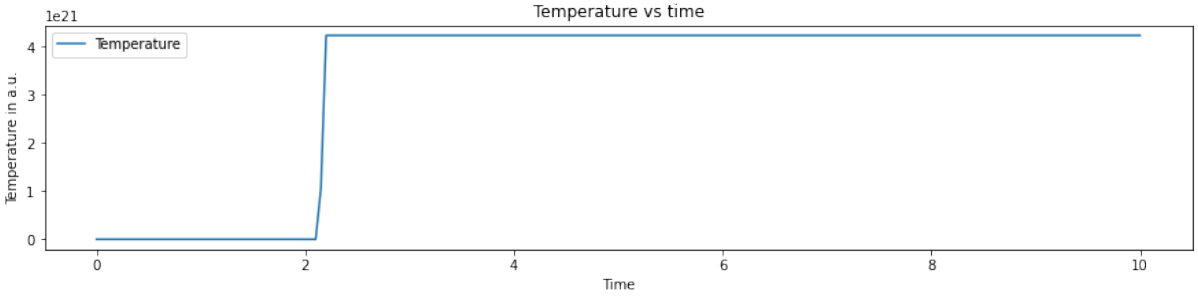


Figure 14: Temperature vs time for $h = 0.05$.

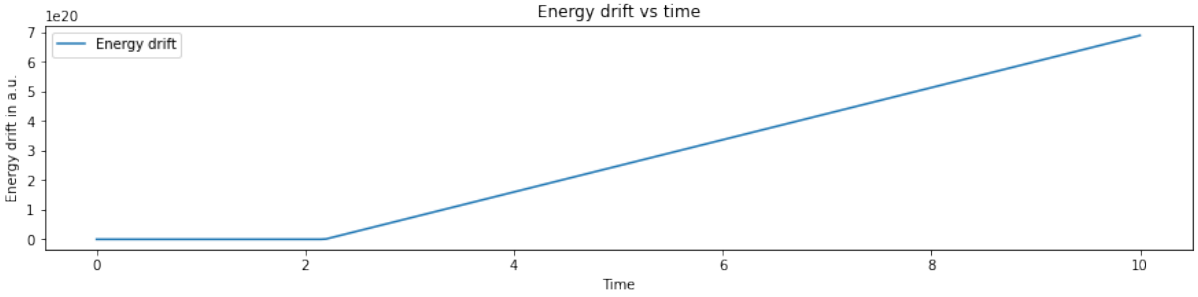


Figure 15: Energy drift for $h = 0.05$.

### B.1.6 H=0.1

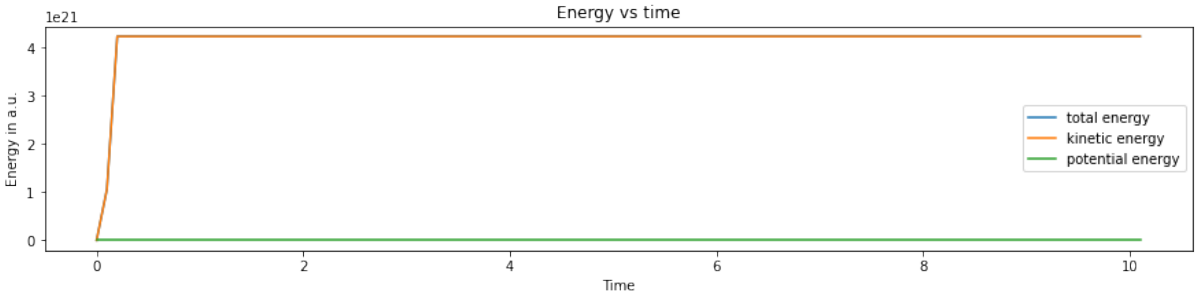


Figure 16: Total, potential and kinetic energy vs time for $h = 0.1$.

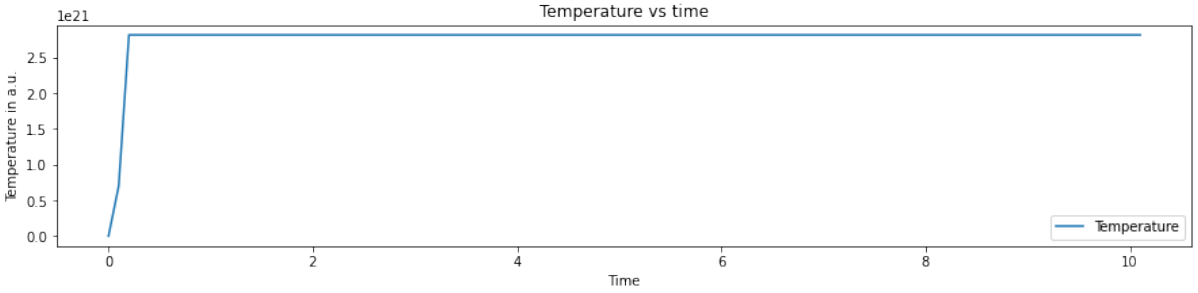


Figure 17: Temperature vs time for $h = 0.1$.

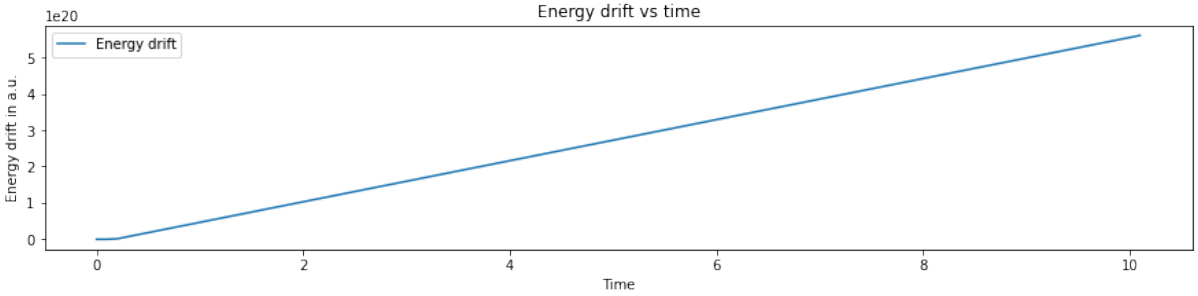


Figure 18: Energy drift for $h = 0.1$.

### B.1.7 T=5

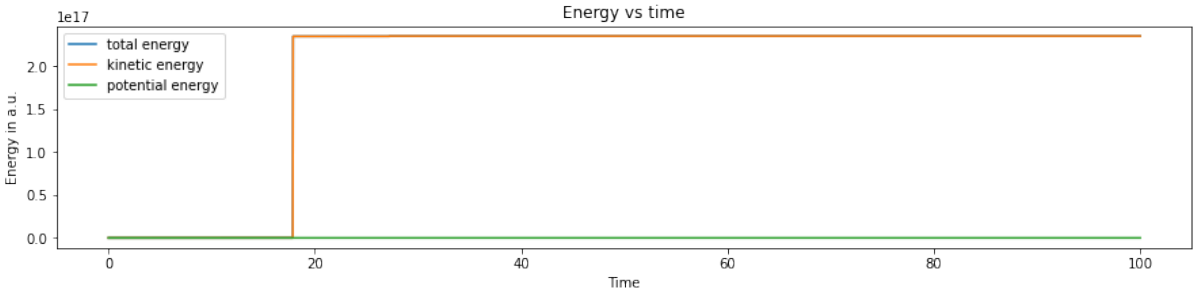


Figure 19: Total, potential and kinetic energy vs time for $t = 5$.

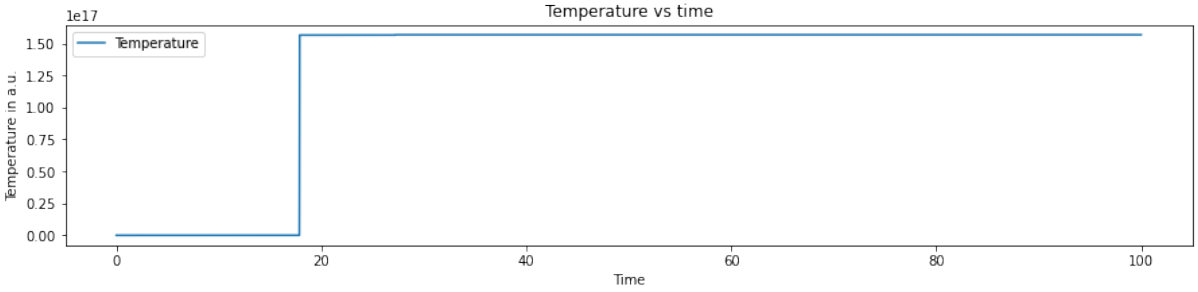


Figure 20: Temperature vs time for $t = 5$.

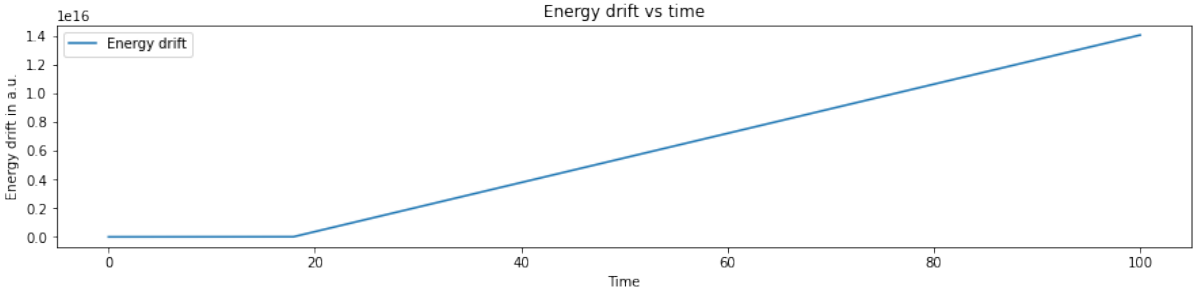


Figure 21: Energy drift for $t = 5$.

### B.1.8 T=7

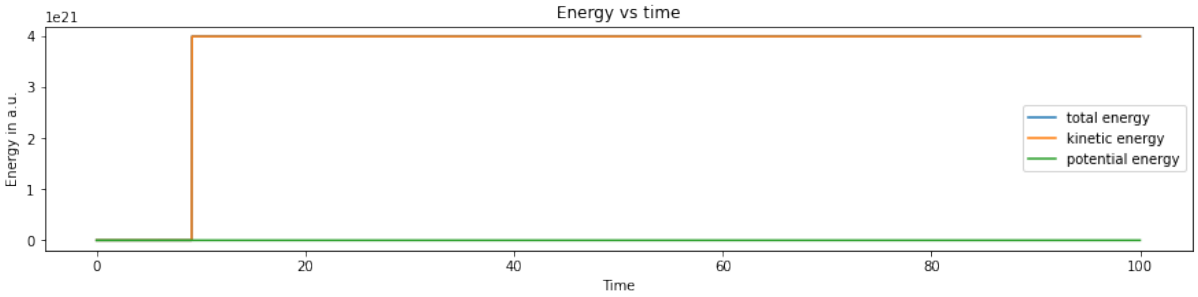


Figure 22: Total, potential and kinetic energy vs time for $t = 7$.

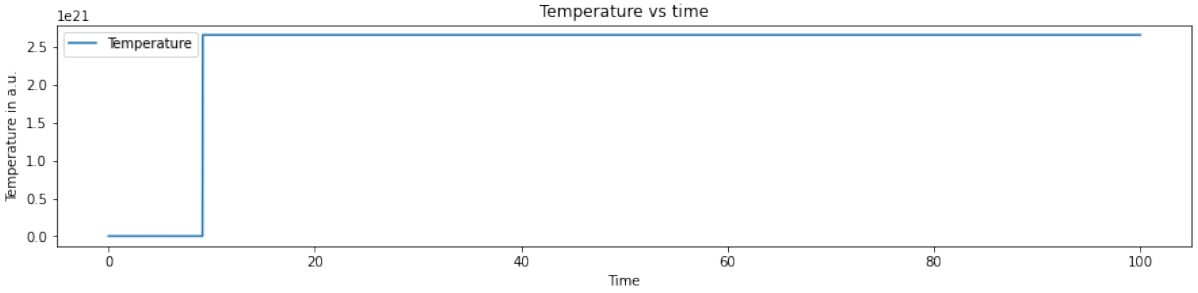


Figure 23: Temperature vs time for $t = 7$.

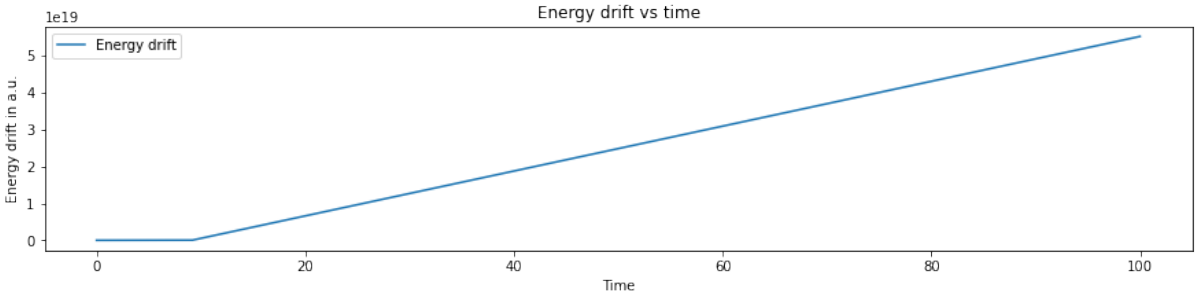


Figure 24: Energy drift for $t = 7$.

### B.1.9 T=9

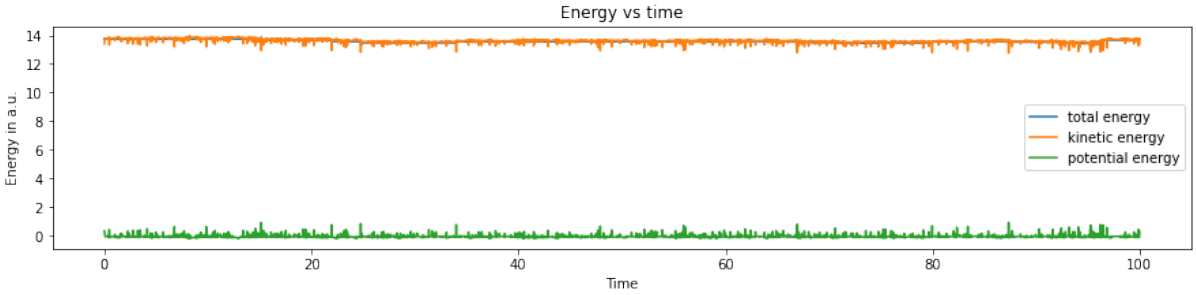


Figure 25: Total, potential and kinetic energy vs time for $t = 9$.

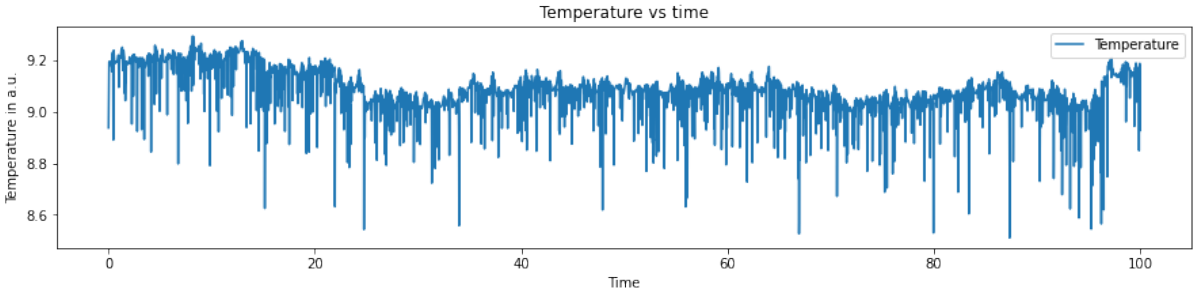


Figure 26: Temperature vs time for $t = 9$.

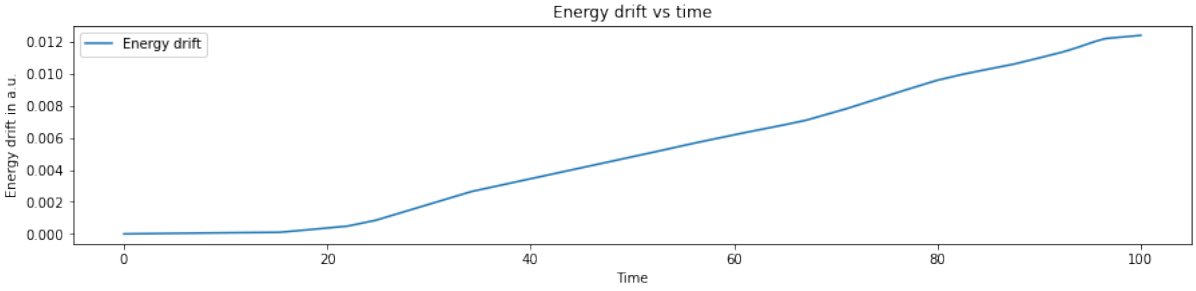


Figure 27: Energy drift for $t = 9$.

### B.1.10 T=11

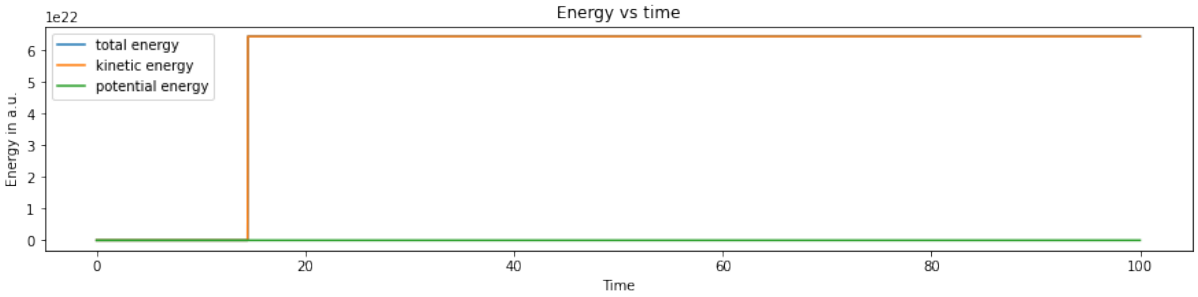


Figure 28: Total, potential and kinetic energy vs time for $t = 11$.

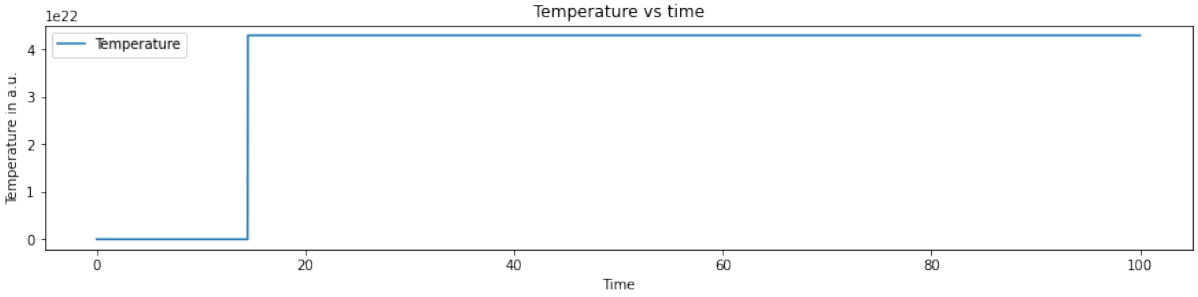


Figure 29: Temperature vs time for $t = 11$.

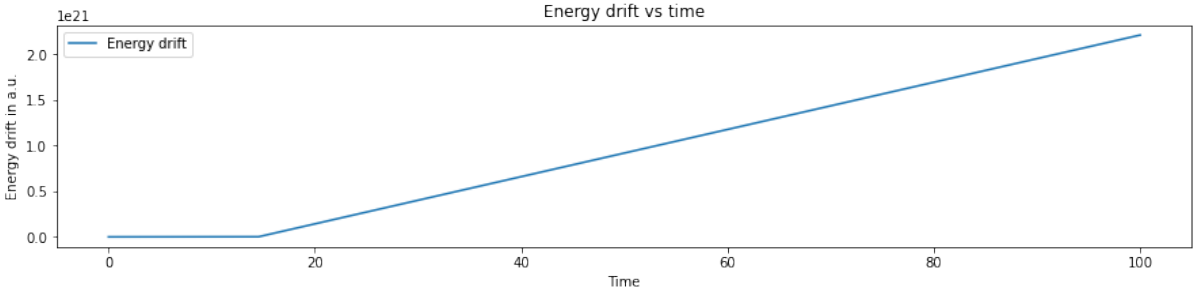


Figure 30: Energy drift for $t = 11$.

### B.1.11 Different Values of N

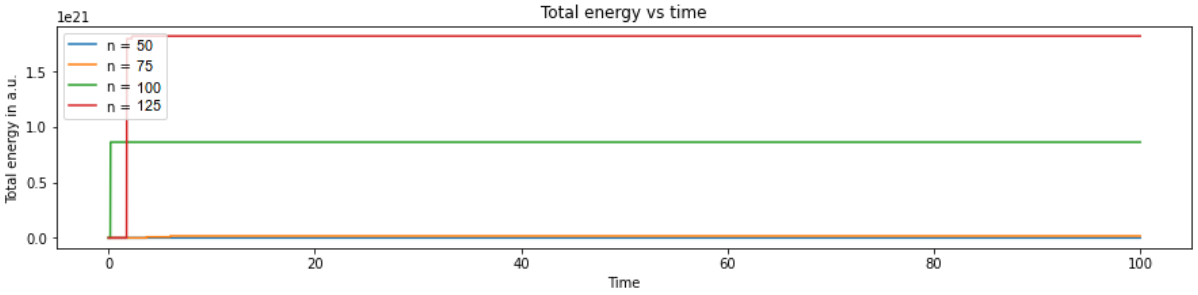


Figure 31: Total energy vs time for different $N$ values.

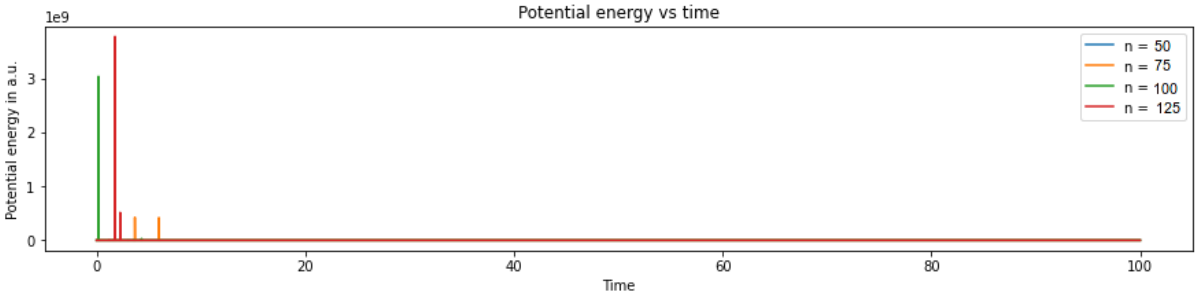


Figure 32: Potential energy vs time for different $N$ values.

## B.2 Assignment 3

### B.2.1 Mean Squared Displacement

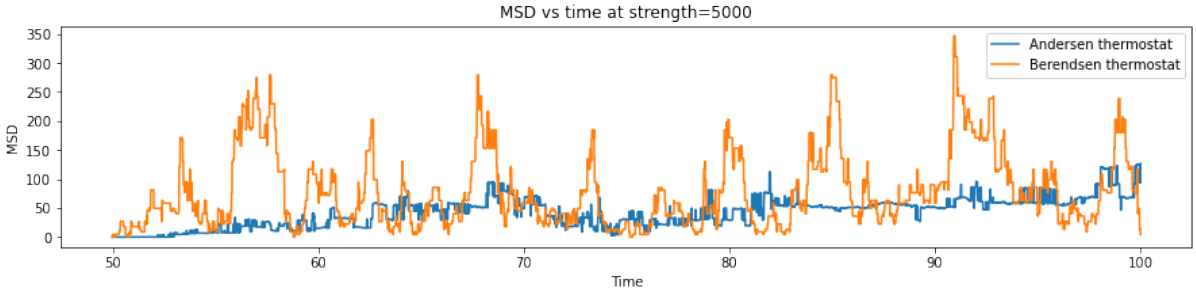


Figure 33: Mean squared displacement at a thermostat strength of 5000.

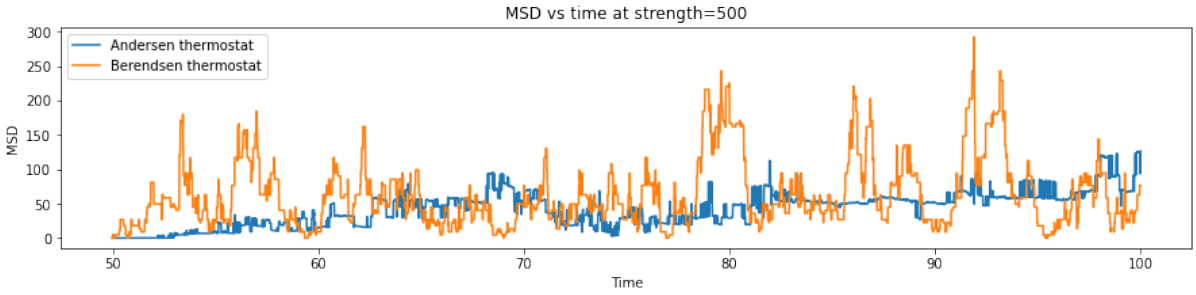


Figure 34: Mean squared displacement at a thermostat strength of 500.

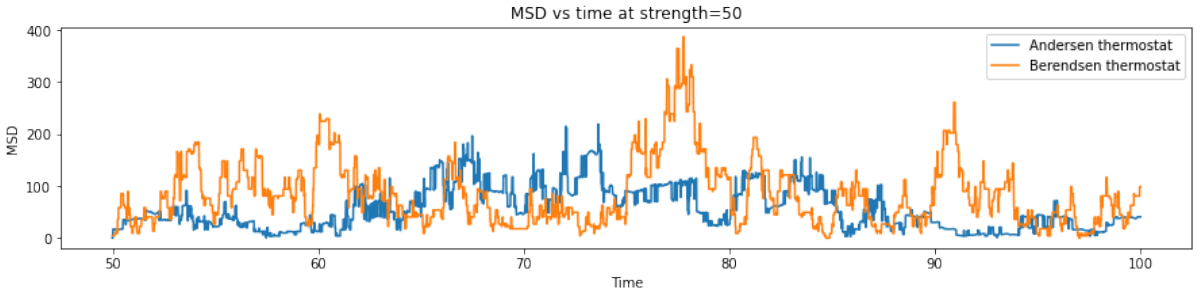


Figure 35: Mean squared displacement at a thermostat strength of 50.

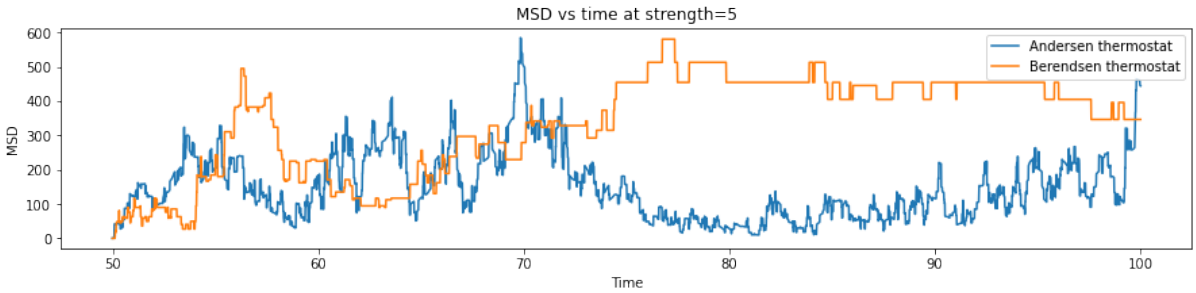


Figure 36: Mean squared displacement at a thermostat strength of 5.

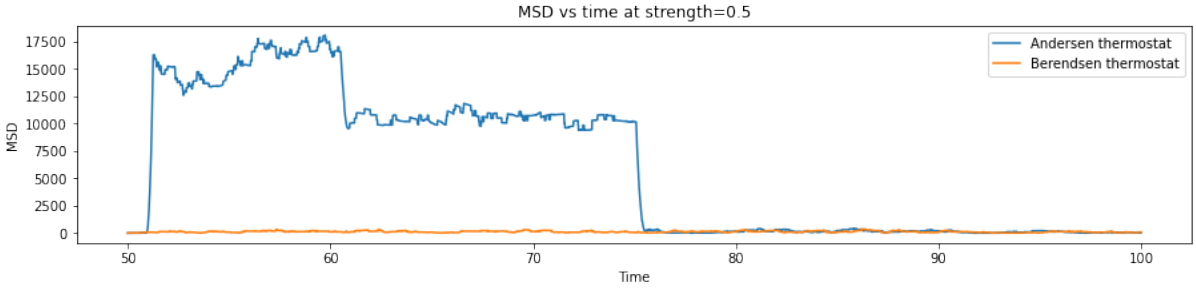


Figure 37: Mean squared displacement at a thermostat strength of 0.5.

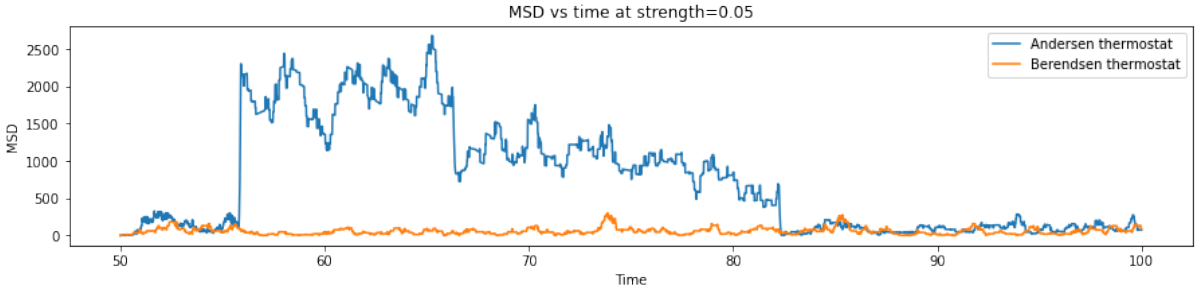


Figure 38: Mean squared displacement at a thermostat strength of 0.05.

### B.2.2 Andersen Thermostat Velocity Distributions

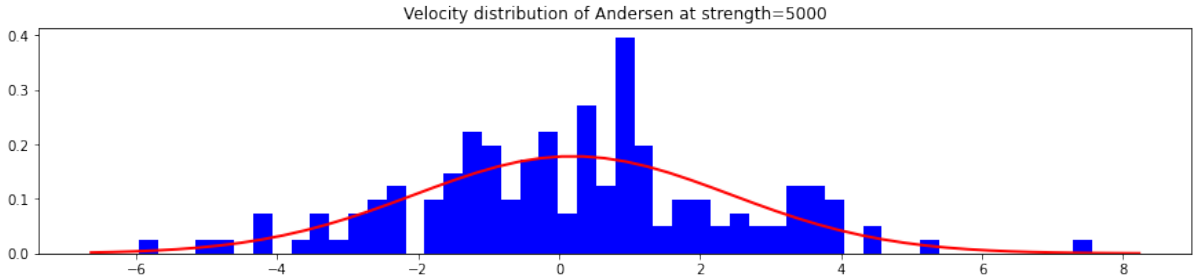


Figure 39: Velocity distribution for the Andersen thermostat at a thermostat strength of 5000.

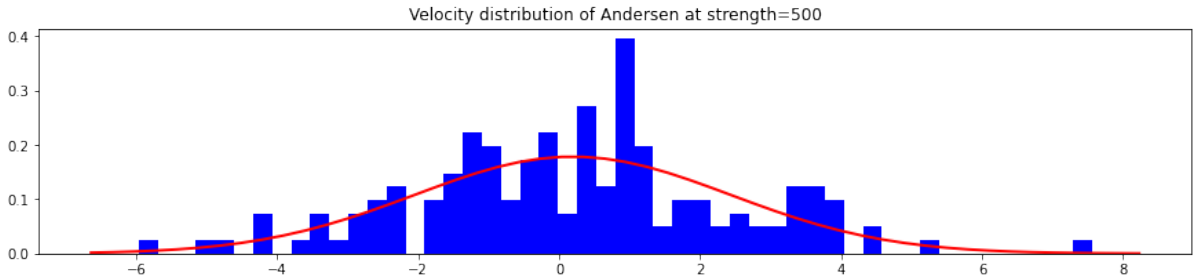


Figure 40: Velocity distribution for the Andersen thermostat at a thermostat strength of 500.

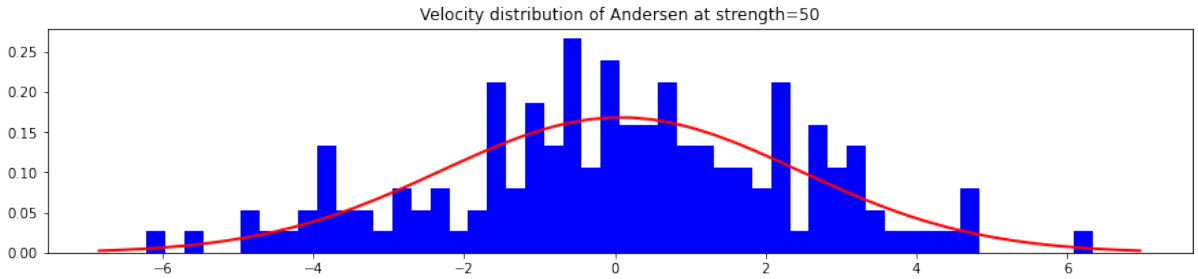


Figure 41: Velocity distribution for the Andersen thermostat at a thermostat strength of 50.

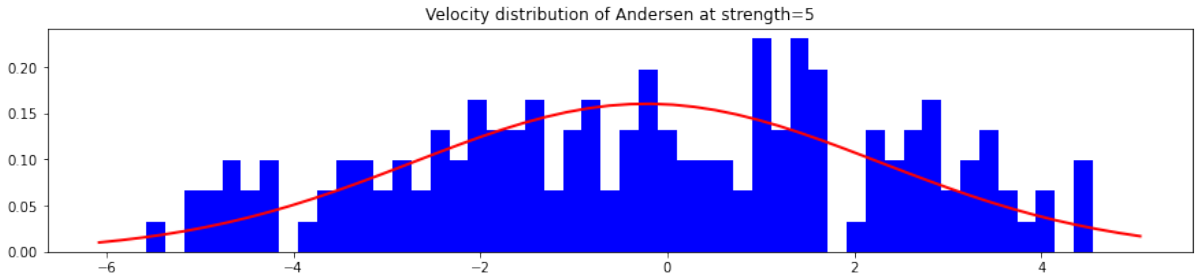


Figure 42: Velocity distribution for the Andersen thermostat at a thermostat strength of 5.

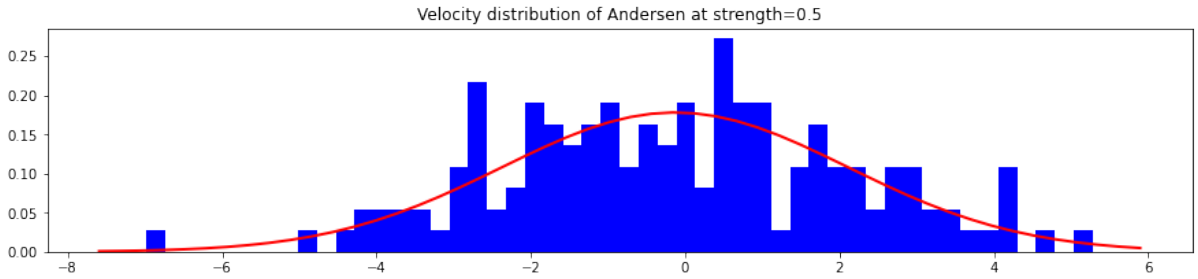


Figure 43: Velocity distribution for the Andersen thermostat at a thermostat strength of 0.5.

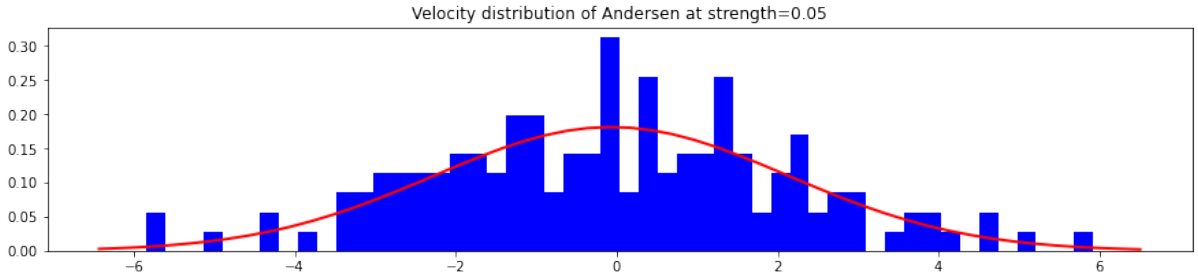


Figure 44: Velocity distribution for the Andersen thermostat at a thermostat strength of 0.05.

### B.2.3 Berendsen Thermostat Velocity Distributions

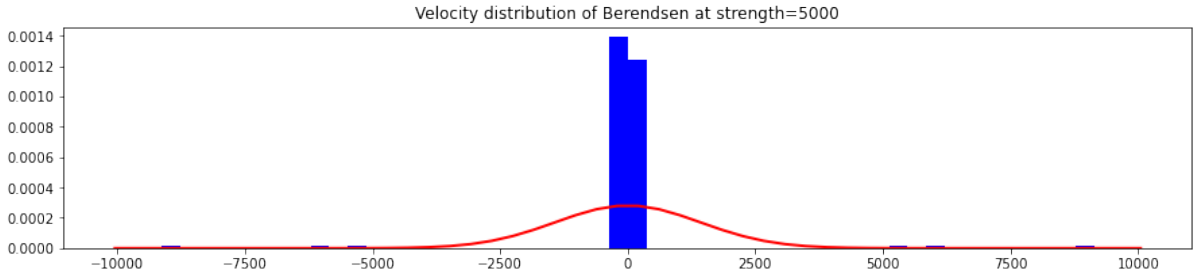


Figure 45: Velocity distribution for the Berendsen thermostat at a thermostat strength of 5000.

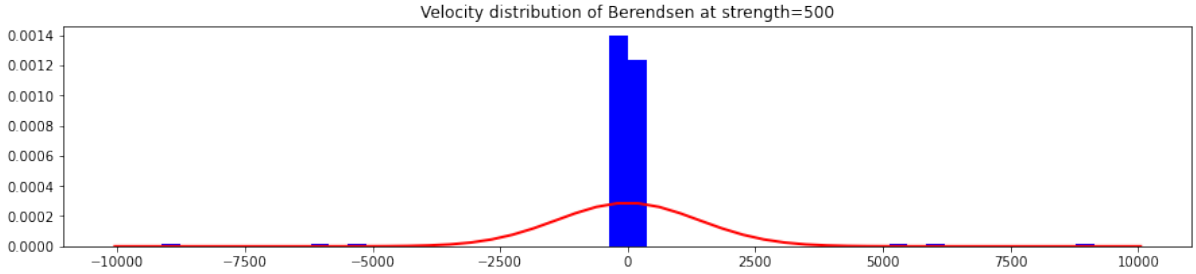


Figure 46: Velocity distribution for the Berendsen thermostat at a thermostat strength of 500.

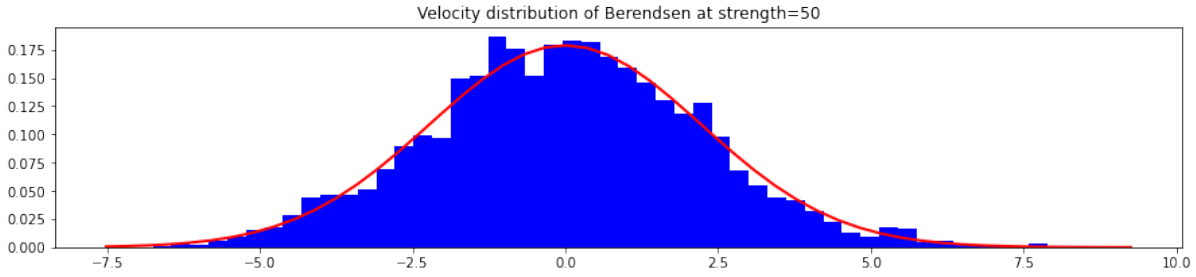


Figure 47: Velocity distribution for the Berendsen thermostat at a thermostat strength of 50.

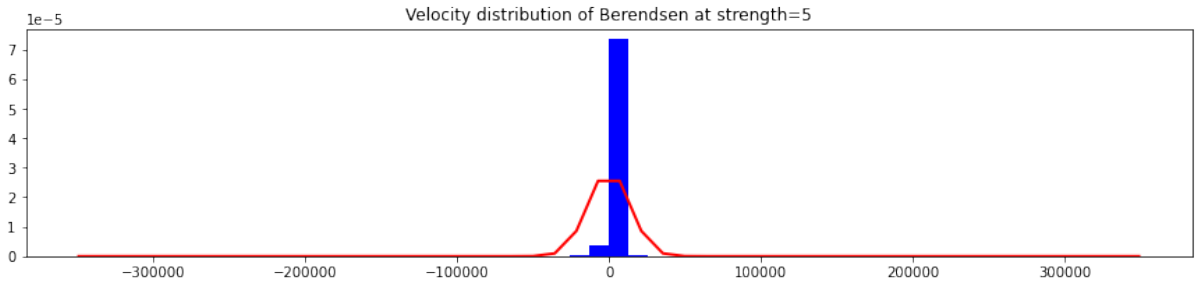


Figure 48: Velocity distribution for the Berendsen thermostat at a thermostat strength of 5.

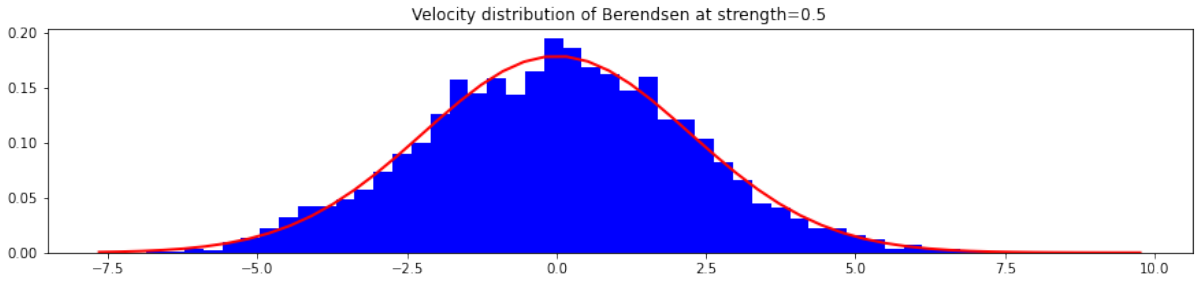


Figure 49: Velocity distribution for the Berendsen thermostat at a thermostat strength of 0.5.

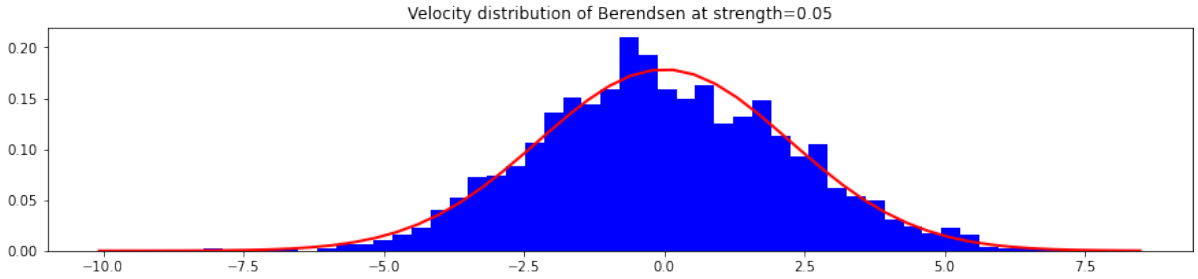


Figure 50: Velocity distribution for the Berendsen thermostat at a thermostat strength of 0.05.

### B.2.4 Pressure Calculation

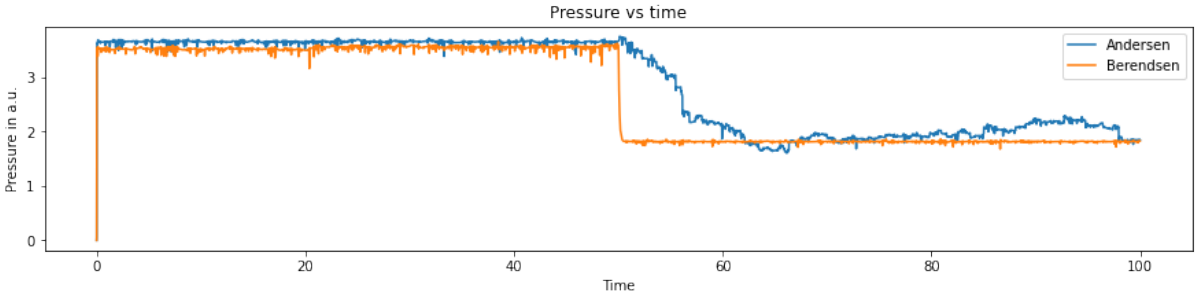


Figure 51: Pressure vs time in a system where the Andersen thermostat is turned on half-way.

### B.2.5 Berendsen Barostat

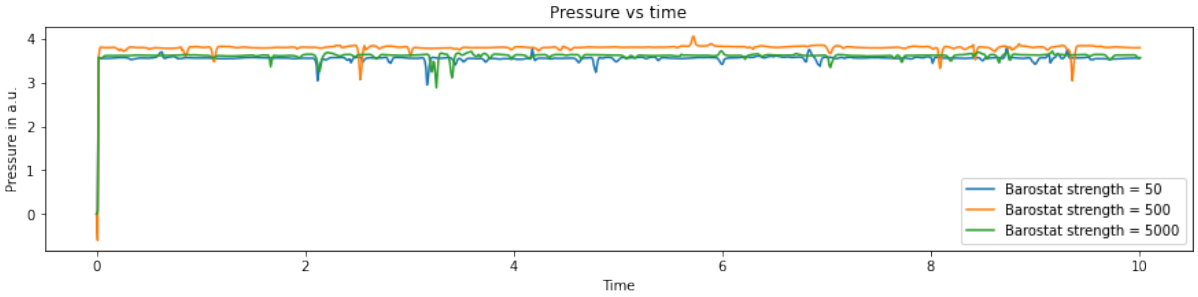


Figure 52: Pressure vs time for a simulation in which the Berendsen barostat is enabled half-way.

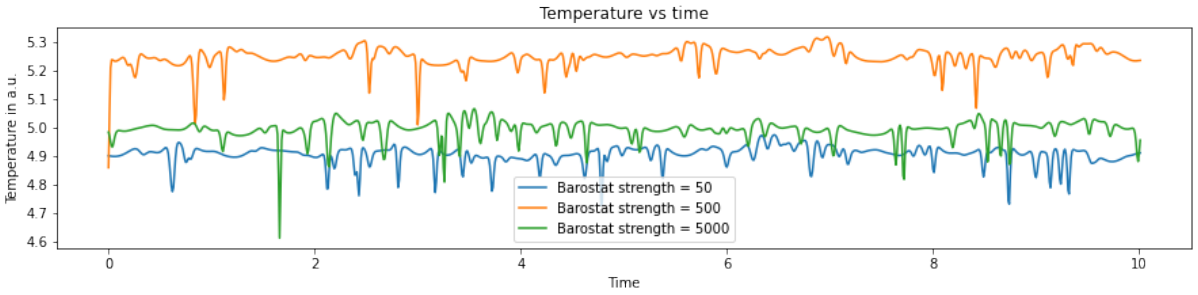


Figure 53: Temperature vs time for a simulation in which the Berendsen barostat is enabled half-way.