

Exam PUE Advanced Computational Physics

Jonas Bojanovsky, 12113264

June 18, 2025

Abstract

This is my submission of the final PUE exam. You can find supporting data and information about how exactly the code was implemented in the supporting Jupyter Notebook. That, however, is very raw and not nice to read, and this document should be mostly complete or at least consistent. Note that unless explicitly SI units are used, $k_B = 1$, so $k_B T$ is just written as T , for example.

1 Units and periodic boundary conditions

We consider a Lennard Jones system with three particles. We have a quadratic box with side length $L = 12 \text{ \AA}$

The three particles are at the positions:

$$\vec{r}_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \text{ \AA} \quad \vec{r}_2 = \begin{pmatrix} 1 \\ 1 \\ 11 \end{pmatrix} \text{ \AA} \quad \vec{r}_3 = \begin{pmatrix} 11 \\ 11 \\ 11 \end{pmatrix} \text{ \AA} \quad (1)$$

Under the minimum image convention, we can implement a simple code to calculate the distances between the particles, implemented by the code

```
1 for i in range(3):
2     if ri[i] - rj[i] > L / 2:
3         rj[i] += L
4     elif ri[i] - rj[i] < -L / 2:
5         rj[i] -= L
6     return rj - ri
```

Listing 1: Implementing Minimum image convention

Then, using \vec{r}_1 as the reference vector, we can find the center of mass in the minimum image convention, \vec{X} , by adding the MIC distances to \vec{r}_1 and averaging. This yields

$$\vec{X} = \frac{1}{3} \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} \text{ \AA} \quad (2)$$

Note that if we need to wrap it back into the original box, naturally this would become $(1/3 \ 1/3 \ 35/3)^T$. With $\sigma = 3.627 \text{ \AA}$, the reduced center of mass is $\vec{X}^* = \vec{X}/\sigma$:

$$\vec{X}^* = 0.0919 \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}. \quad (3)$$

And in the scaled coordinates, we instead scale by the length: $\vec{X}' = \vec{X}/L$:

$$\vec{X}' = 0.0277 \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}. \quad (4)$$

Calculating the potential energy, we use the contributions only from the three particles under the minimum image convention and do not consider the ones from other cells.

$$U_{12} = 4919.11\varepsilon \quad (5)$$

$$U_{13} = 1.67\varepsilon \quad (6)$$

$$U_{23} = 61.30\varepsilon \quad (7)$$

The total sum of these contributions is

$$U = 4982.09\varepsilon.$$

2 Parallel tempering MC simulation

2.1 Detailed balance for parallel tempering step

A typical acceptance rule in the Metropolis step to preserve detailed balance is

$$p_{\text{acc}}(x \rightarrow y) = \min \left[1, \frac{f(y)}{f(x)} \right], \quad (8)$$

where y (x) is the new (old) sample from the ensemble and f the corresponding probability density function. For our case,

$$f(x) = \frac{1}{Q_{\text{extended}}} \exp[-\beta_i U(x_i)] \exp[-\beta_j U(x_j)] \prod_{k \neq i, j} \exp[-\beta_k U(x_k)] \quad (9)$$

In the trial step, we want to try to swap the positions from the trajectories from two different temperatures, T_i and T_j . So the swapped probability is:

$$f(y) = \frac{1}{Q_{\text{extended}}} \exp[-\beta_j U(x_i)] \exp[-\beta_i U(x_j)] \prod_{k \neq i, j} \exp[-\beta_k U(x_k)] \quad (10)$$

The ratio simplifies:

$$\frac{f(y)}{f(x)} = \exp[(\beta_i - \beta_j)(U(x_i) - U(x_j))] = \exp[\Delta\beta\Delta U], \quad (11)$$

hence we have arrived at our desired acceptance probability.

2.2 Implementing the code

The implementation of the potential can be seen in Appendix A. The implementation of the MC with parallel tempering is in Appendix B. The distributions of the trajectories with and without parallel tempering are in the figures 1 and 2 respectively.

The trajectory of the $T = 0.05$ particle with tempering at different zoom levels can be seen in Figures 3 – 5. For comparison, the position vs. time graph of the no-tempering particle can be seen in Figure 6.

The simulation was run for 1×10^6 steps. The acceptance ratio can be seen in Table 1.

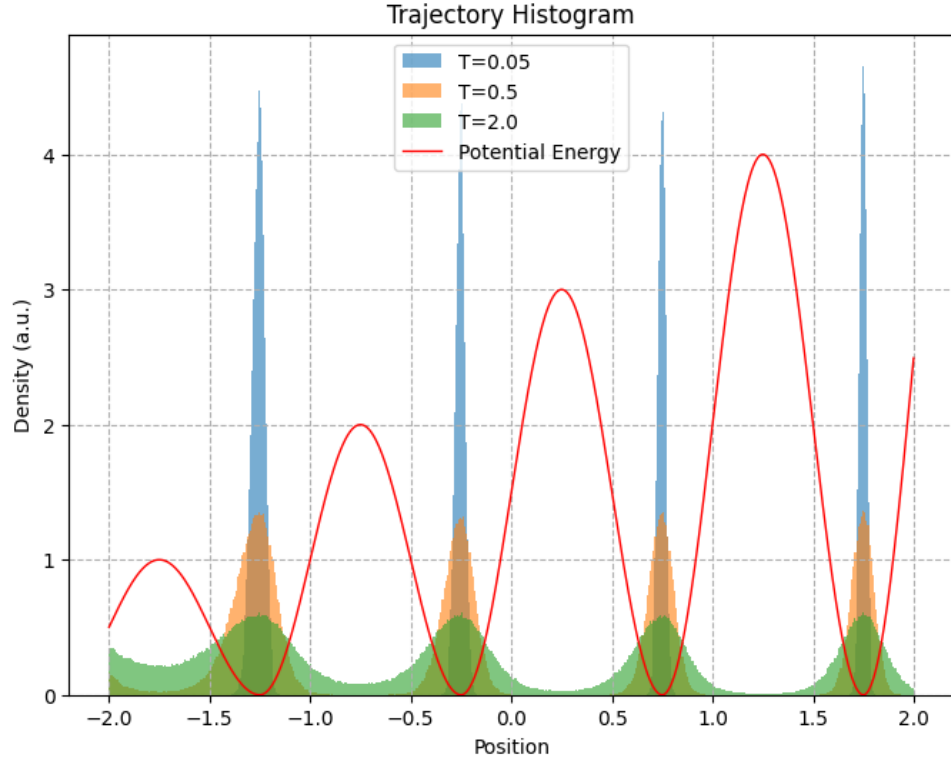


Figure 1: Histogram of the particle position at different temperatures **with parallel tempering**. Potential only for visualization purposes, arbitrary units.

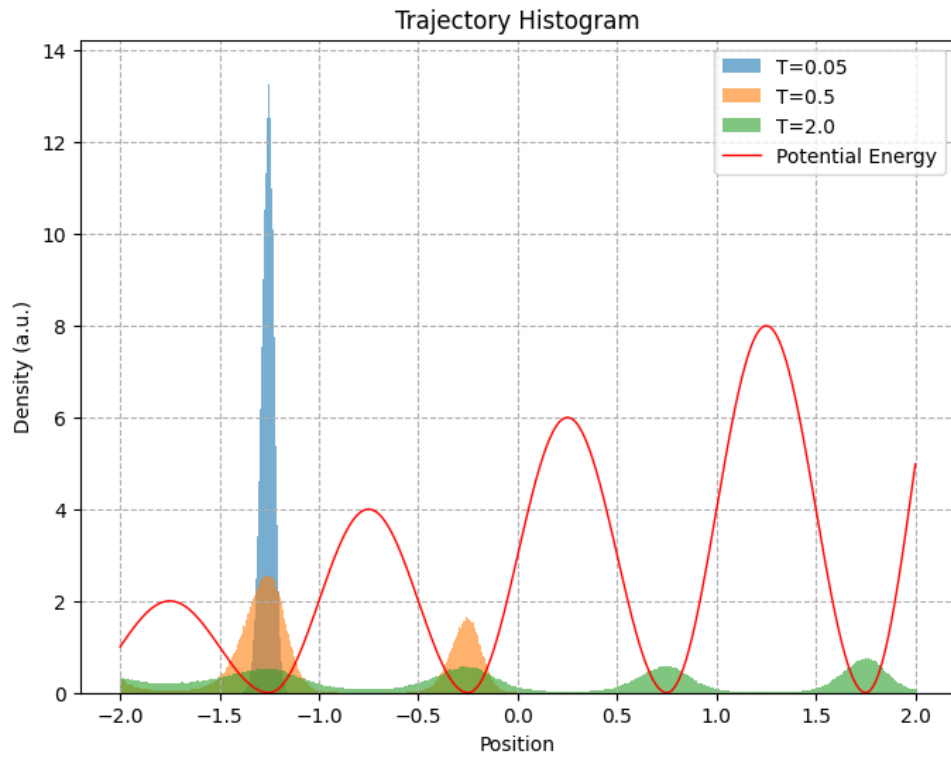


Figure 2: Histogram of the particle position at different temperatures **without parallel tempering**. Potential only for visualization purposes, arbitrary units.

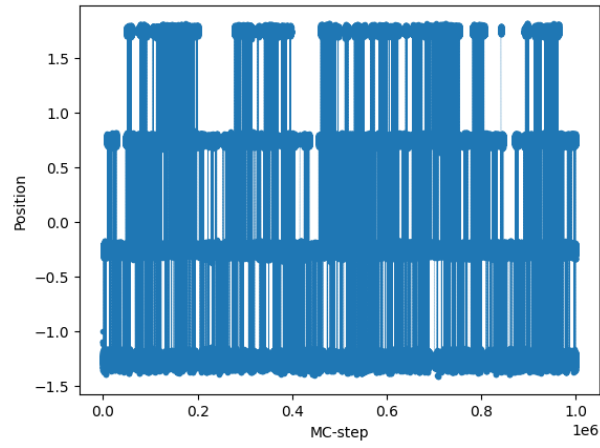


Figure 3: position of $T = 0.05$ particle against time

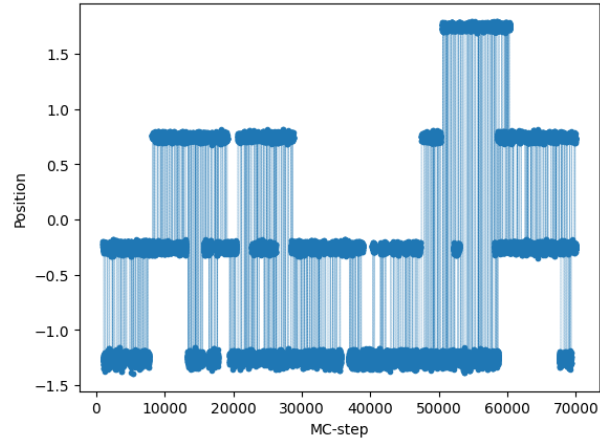


Figure 4: position of $T = 0.05$ particle against time

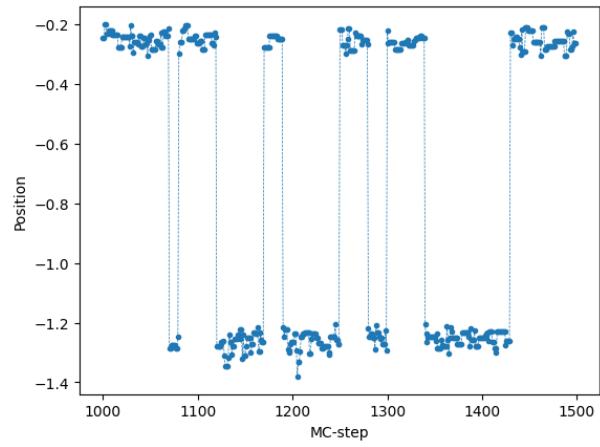


Figure 5: position of $T = 0.05$ particle against time

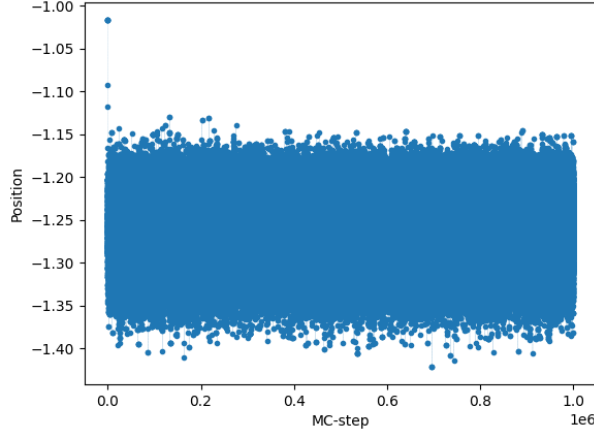


Figure 6: position of $T = 0.05$ particle against time, no tempering

Table 1: Acceptance ratios of MC simulation with and without parallel tempering (bottom two lines are the swapping acceptances for the tempering step)

| T | tempering | no temp. |
|----------------------------|-----------|----------|
| 0.05 | 0.37 | 0.46 |
| 0.5 | 0.74 | 0.79 |
| 0.2 | 0.88 | 0.89 |
| $0.05 \leftrightarrow 0.5$ | 0.37 | — |
| $0.5 \leftrightarrow 2.0$ | 0.53 | — |

2.3 Block averaging

When estimating the standard error in the block averaging method, one tries to correct for the correlation in the samples by dividing the ‘trajectory’ into blocks of size B and calculating the variance of these averages. Note that there are $M = \lfloor N/B \rfloor$ blocks of this size, where N is the number of simulation steps (in our case 1 Million). Then the estimate for the true variance of the mean is

$$(\Delta \bar{x})^2 = \frac{1}{M(M-1)} \sum_{j=1}^M (\bar{X}_j - \langle x \rangle)^2, \quad (12)$$

where $\langle x \rangle$ is the mean of the position (independent of the block size), and \bar{X}_j is the position average of the j^{th} block.

To find the best possible estimate, one should calculate this for different block sizes and see where the computed value converges, while keeping the number of blocks M large enough so that statistics is still meaningful. In this simulation, this was not easy at all because sometimes, the values did not seem to converge at all, see Figure 7.

Nevertheless, the estimated values can be found in Table 2. As a sanity check, we ran the simulation 10 times with tempering and recorded the average position each time. Then, computing the standard deviation of these averages should give a good estimate of the standard error, which is the deviation of the mean. The results here agree well, as we get about $\Delta \bar{x}_{\text{ann.}} = 0.08$ for all temperatures, agreeing well with the estimate from the block-averaging method. The values for the standard error in Table 2 was extracted by simply looking at the graphs and looking at convergence/local maxima and rounding up visually.

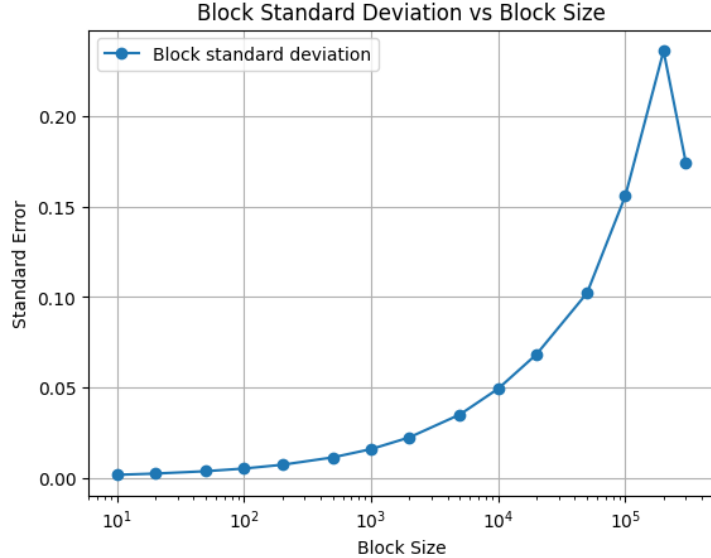


Figure 7: Standard error scaling with block size for $N = 1e6$ steps, no annealing, $T = 0.5$.

Table 2: Estimates for position averages and Standard errors with tempering (subscript ‘temp.’) and without.

| T | $\langle x \rangle_{\text{temp.}}$ | $\Delta \bar{x}_{\text{ann.}}$ | $\langle x \rangle$ | $\Delta \bar{x}$ |
|------|------------------------------------|--------------------------------|---------------------|--------------------|
| 0.05 | 0.07 | 0.08 | -1.25 | 7×10^{-5} |
| 0.5 | 0.02 | 0.07 | -0.96 | 0.25 |
| 0.2 | -0.09 | 0.07 | -0.16 | 0.09 |

2.4 Discussion

As expected from the histograms, the particles spend most of their ‘time’ in the low-potential regions. Parallel tempering seems to be very effective to let the low-temperature trajectories explore local minima other than the one at -1.25, which is the one the system naturally falls into most likely due to the initial position $x = -1$. Interestingly enough, in the tempering case, the particle spent slightly more time in the right-most minimum than the more left ones (this varied from run to run), probably because once it got there, due to the high maximum at $x = 1.25$, it could not get back so easily.

In the trajectories vs. time figures, one can clearly see how the particle jumps between the minima, showing the tempering process. In Figure 6, where there is no tempering – and especially at low temperature – the particle just stays in the first minimum it falls into until the end of the simulation.

Note that at the beginning of the simulation there was an equilibration time, but after 100 steps the system was already equilibrated, so that this was sometimes just neglected, and sometimes the first 100 steps were thrown away when computing values, especially the averages.

The acceptance ratios of the MC steps get higher as the temperature increases – as expected. Interestingly, for $T = 0.05$, the tempering case is significantly lower than the no-tempering case. This is likely due to the fact that, when swapping with the higher-temperature particle, there are more ‘forbidden’ regions, and the particle in the no-tempering case just sits in its local minimum comfortably, which is locally relatively flat.

In the tempering case, the $\langle x \rangle$ move to the left with higher temperature. This is probably due to the minimum at -2, which become more accessible to higher-temperature particles, but the other ones at $U = 0$ potential are due to tempering equally accessible to all particles. On the other hand, the no-tempering averages move to the right with higher temperature, because

only the higher-temperature particles ever overcome the higher and higher barriers to the right.

An outlier in the standard error is the no-tempering, $T = 0.05$ case. Because it very securely sits in the -1.25 minimum, it is extremely well localized there, hence the low $\Delta\bar{x}$.

3 Exponential correlations

From the ACOMP script we know that $\text{MSD} \equiv \langle [r(t) - r(0)]^2 \rangle$ is related to the VACF by

$$\text{MSD} = 2 \cdot \int_0^t dt' \int_0^{t'} dt'' \langle v(0)v(t'') \rangle \quad (13)$$

$$= 2 \cdot \int_0^t dt' \langle v^2 \rangle \tau \left(1 - e^{-t'/\tau}\right) \quad (14)$$

$$= 2 \langle v^2 \rangle \tau \left[t - \tau \left(1 - e^{-t/\tau}\right) \right] \quad (15)$$

Similarly, we can get the diffusion coefficient D :

$$D = \frac{1}{3} \int_0^\infty dt \langle v(0)v(t) \rangle \quad (16)$$

$$\Rightarrow D = \frac{1}{3} \langle v^2 \rangle \tau \quad (17)$$

Further, if we expand the MSD, we get

$$2 \langle v^2 \rangle \left[\tau t - \tau^2 \left(1 - 1 - \frac{t}{\tau} - \frac{t^2}{2\tau^2} - O(t^3)\right) \right] \quad (18)$$

$$= \langle v^2 \rangle t^2 + O(t^3). \quad (19)$$

So for small t , the system behaves ballistically (MSD scales with t^2). On the other hand, for $t \rightarrow \infty$ (same as $\tau \rightarrow 0$), we get

$$\text{MSD} \rightarrow 2 \langle v^2 \rangle \tau t = 6Dt, \quad (20)$$

which is consistent with theory, and is diffusive motion. So τ is like a relaxation or diffusion time, how long the system ‘remembers’ its original velocity, i.e. stays in the ballistic regime before transitioning to the diffusive one.

For water, we assume $D = 31 \times 10^{-5} \text{ cm}^2/\text{s}$. From the Boltzmann distribution we know that

$$\langle v^2 \rangle = \frac{3k_B T}{m}, \quad (21)$$

where $m \approx 18 \text{ amu} = 3 \times 10^{-26} \text{ kg}$. From that we get $\langle v^2 \rangle = 4.16 \times 10^9 \text{ cm}^2/\text{s}$. Now, using Equation 17, we get

$$\tau = 22 \text{ fs}.$$

4 Judging results

1. Energy in the NVE ensemble should be (almost) perfectly constant, but here it increases.
2. Sudden step in potential energy hints at a melting process, or at least some breaking of the FCC structure, which is completely ignored. This would be consistent with the parameters, because at $T = 1.5$ and $\rho^* = 0.8$ there should be gas phase for an LJ system, at least definitely not solid.
3. There is no plot of the kinetic energy, why?

4. There is no mention of the time step used, which is critical to examine stability, especially for integrators like Euler-forward.

5. Euler forward is a really bad integrator for energy conserving simulations. A symplectic integrator like velocity-verlet should be implemented instead.

6. 30 particles really is not many for such a system and finite-size effects may play a role, but it is completely ignored.

7. The temperature plot is completely missing from this, although it should not be expected to stay constant.

8. Is there a cutoff radius in the simulation? We use periodic boundary conditions. The cutoff radius may also be difficult.

Based on these issues, especially the ignoring of the melting, missing information about the time step and bad choice of integrator, I would not trust these results or the speaker that they really knew what they were doing.

Appendices

A Defining Potential and quantities

```
1 def U(x):
2     """Potential"""
3     U = 1 + np.sin(2 * np.pi * x)
4
5     if x < -2 or x > 2:
6         return np.inf
7     elif -1.25 <= x <= -0.25:
8         U *= 2
9     elif -0.25 <= x <= 0.75:
10        U *= 3
11    elif 0.75 <= x <= 1.75:
12        U *= 4
13    elif 1.75 <= x <= 2:
14        U *= 5
15    return U
16
17
18    Temperatures = np.array([0.05, 0.5, 2.0]) # k_B = 1
19    betas = 1/Temperatures
20
21    x_initial = -1.0
22
23    maximum_displacement = 0.1
24
25    parallel_tempering_after = 10
```

Listing 2: Defining the potential and fixed quantities

B Metropolis MC algorithm with parallel tempering

```
1 def MC_step(x, beta, displacement):
2     """Perform a single Monte Carlo step based of precalculated displacement
3
4     Args:
5         x (float): Current position
6         T (float): Temperature
7         displacement (float): Displacement to apply
8
9     Returns:
10        tuple: New position and whether the move was accepted
11    """
12    x_new = x + displacement
13    U_x = U(x)
14    U_x_new = U(x_new)
15
16    if U_x_new < U_x:
17        return x_new, True
18    else:
19        p_accept = np.exp(-(U_x_new - U_x) * beta)
20        if np.random.rand() < p_accept:
21            return x_new, True
22        else:
23            return x, False
24
25
```

```

26
27 def Parallel_Tempering_step(x1, x2, beta1, beta2):
28     """Perform a parallel tempering step between two temperatures by swapping
29     positions if accepted
30
31     Args:
32         x1 (float): Current position at temperature T1
33         x2 (float): Current position at temperature T2
34         beta1 (float): Inverse temperature 1 (1/T1)
35         beta2 (float): Inverse temperature 2 (1/T2)
36
37     Returns:
38         whether the swap was accepted
39     """
40     U_x1 = U(x1)
41     U_x2 = U(x2)
42
43
44     p_swap = np.exp((beta1 - beta2) * (U_x1 - U_x2))
45     if np.random.rand() < p_swap:
46         return x2, x1, True
47     else:
48         return x1, x2, False
49
50
51 def precalculate_displacements(maximum_displacement, n_steps):
52     """Precalculate displacements for the Monte Carlo steps
53
54     Args:
55         maximum_displacement (float): Maximum displacement allowed
56         n_steps (int): Number of steps to precalculate
57
58     Returns:
59         np.ndarray: Array of displacements
60     """
61     return np.random.rand(n_steps) * maximum_displacement * 2 -
62         maximum_displacement
63
64 def simulation(n_steps, betas = betas, x_initial = x_initial,
65     parallel_tempering_after = parallel_tempering_after, maximum_displacement =
66     maximum_displacement, parallel_tempering = True):
67     """Run the Monte Carlo simulation"""
68
69     n_temperatures = len(betas)
70     displacements = [precalculate_displacements(maximum_displacement, n_steps)
71         for i in range(n_temperatures)]
72
73     x = np.full(n_temperatures, x_initial)
74     energies = np.zeros((n_temperatures, n_steps))
75     mc_acceptance_ratio = np.zeros(n_temperatures)
76     swap_acceptance_ratio = np.zeros((n_temperatures, n_temperatures))
77
78     trajectories = np.empty((n_temperatures, n_steps))
79
80     for step in range(n_steps):
81         for i in range(n_temperatures):
82             x_new, accepted = MC_step(x[i], betas[i], displacements[i][step])
83             energies[i, step] = U(x_new)
84             if accepted:
85                 x[i] = x_new
86                 mc_acceptance_ratio[i] += 1

```

```

84         if step % parallel_tempering_after == 0 and step > 0 and
           parallel_tempering:
85             # Perform parallel tempering step
86             for i in range(n_temperatures - 1):
87                 x[i], x[i + 1], swap_accepted = Parallel_Tempering_step(x[i], x[
                    i + 1], betas[i], betas[i + 1])
88                 swap_acceptance_ratio[i, i + 1] += swap_accepted
89                 swap_acceptance_ratio[i + 1, i] += swap_accepted
90
91             trajectories[:,step] = x # Store the trajectory of the first
           temperature
92
93     mc_acceptance_ratio /= n_steps
94     swap_acceptance_ratio /= (n_steps // parallel_tempering_after)
95
96     return energies, mc_acceptance_ratio, swap_acceptance_ratio, trajectories
97
98
99 def trajectory_histogram(trajectories, n_bins=100):
100     """Plot histogram of the trajectory
101
102     Args:
103         trajectory (np.ndarray): Trajectory data
104         n_bins (int): Number of bins for the histogram
105     """
106     plt.figure(figsize=(8, 6))
107     for i, trajectory in enumerate(trajectories):
108         plt.hist(trajectory, bins=n_bins, density=True, alpha=0.6, label=f'T={
                Temperatures[i]}')
109
110     # Plot the potential energy for reference
111     x = np.linspace(-2.5, 2.5, 1000)
112     U_x = 0.5*np.vectorize(U)(x)
113     plt.plot(x, U_x, 'r-', lw=1, label='Potential Energy')
114     plt.title('Trajectory Histogram')
115     plt.xlabel('Position')
116     plt.ylabel('Density (a.u.)')
117     plt.grid(linestyle='--')
118     plt.legend()
119     plt.show()

```