

Report for the Course Modelling in Computational Science, HT23

Project 1: The Potts model

Theo Koppenhöfer

Lund

September 16, 2023

```

Input : Initial data  $s, L, T, q$ 
for  $k = 0, 1, \dots$  do
    Pick a point on the grid.
    Propose a new random spin value for this point.
    Calculate the change of energy  $\Delta E$  that this spin flip would cause.
    Accept this spin flip with probability  $\min\{1, \exp(-\Delta E/T)\}$ .
end

```

Algorithm 1: Metropolis

Introduction

The following report is part of the first project of the Modelling in Computational Science course at Lund University, HT2023. In the first project we implemented a Monte-Carlo simulation of the q -state Potts model. The report and the Python implementation can be found online under [1]

The Potts model

The following is a brief introduction to the potts model from statistical mechanics. A state of the q -state Potts model of a $L \times L$ flat grid is given by a mapping

$$s: \{1, \dots, L\} \times \{1, \dots, L\} \rightarrow \{1, \dots, L\}.$$

One assigns this state an energy via

$$E = -J \sum_{\substack{i,j \text{ neighbouring} \\ \text{grid points}}} \delta(s_i, s_j)$$

where δ denotes the Kronecker-delta and $J = 1$ is the coupling strength. As given in the problem setting we assume periodic boundary conditions on the grid.

The algorithms

To model the statistical behaviour of the model we use Monte-Carlo simulations. The first method we implemented was the Metropolis algorithm. The steps for the Metropolis algorithm are given in figure 1. A single iteration of this algorithm is performed by the function `MC_step_fast` in the implementation. In the implementation we used the pseudo-random number generators from `numpy.random` for the proposed spins, spin states and for determining if the choice should be accepted.

Analogously to the Metropolis algorithm the steps of the heat-bath algorithm are given in figure 2. it differs from the Metropolis algorithm only after a point on the grid was randomly chosen. In the implementation a single iteration is performed by the function `Gibbs_step`. In the implementation for this algorithm we also used the `numpy.random` functions.

Input : Initial data s, L, T, q

for $k = 0, 1, \dots$ **do**

Pick a point on the grid.

Pick a spin value for this point with a probability given by the distribution

$$p(s_i) = C \exp\left(\frac{1}{T} \sum_{j \text{ neighbours } i} \delta(s_i, s_j)\right).$$

Here C is a normalisation constant.

end

Algorithm 2: Heat-bath

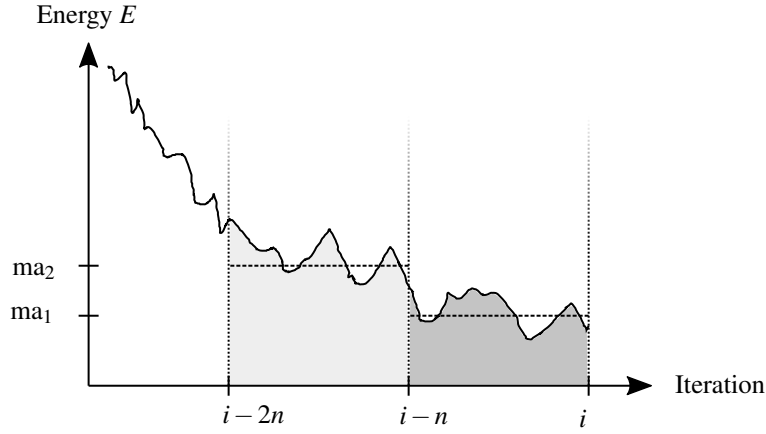


Figure 1: A visualisation of the moving averages.

Some implementation details

Determining when the Energy has plateaued

In order for the simulation to start sampling we needed a criteria to determine the time t_0 when the system reaches an equilibrium state. To determine t_0 we calculate in every step i moving averages ma_1 and ma_2 over n energies. The construction is shown in figure 1. If we start in the hot state then the energy will tend to decrease until we reach an equilibrium. Hence we can use the condition

$$ma_2 \leq ma_1 \tag{1}$$

to determine t_0 . If on the other hand we start in cold state the energy will in general increase and we have to reverse the inequality in equation (1). After reaching an equilibrium the simulation runs for a further `M_sampling` steps. It is over these samples we take the mean and the standard deviation.

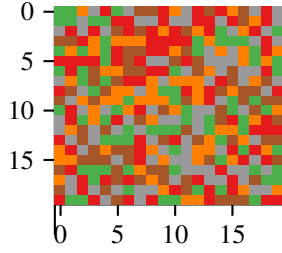


Figure 2: State for temperature $T = 0.1$.

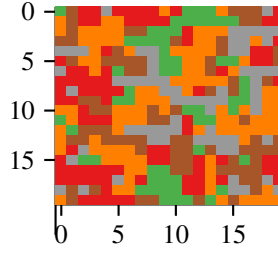


Figure 3: State for temperature $T = 1$.

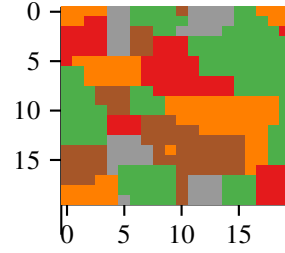


Figure 4: State for temperature $T = 100$.

Improving performance

While simulating we run into performance issues due to the fact that Python is in general rather slow even with heavy use of numpy. We resolved this issue with the help of numba which precompiled functions and thus increased performance substantially. The downside is that the code becomes quite unaesthetic because numba does not support all python and numpy features. In hindsight it would probably been better to have written the iteration in a language other than python. In the experiments we also preferred to use the Metropolis algorithm because our implementation of this algorithm ran faster than the heat-bath algorithm.

The experiments

A brief sanity check

In order to check that our code was indeed doing what it was supposed to we designed some sanity checks. The energy during the simulation is updated with the calculated value for ΔE . Hence we checked with the function `test_energies` if the energy of the end state of the simulation is the same as the energy calculated during the simulation.

We also ran the simulation

State plots

In figures 4 to 2 we can see that as the temperature increases the state of the system becomes more chaotic. For low temperatures there are larger patches of the same state which corresponds to a lower energy configuration. In the implementation one can see an animated version of the evolution of the system state.

Distribution of energies in equilibrium

In the following numerical experiment we set the grid size to $L = 300$, the spin states to $q = 10$ and the temperature to $T = 10^2$. We then run the simulation until the system

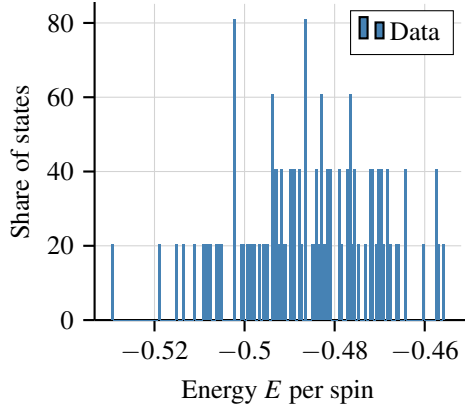


Figure 5: For $M = 10^5$

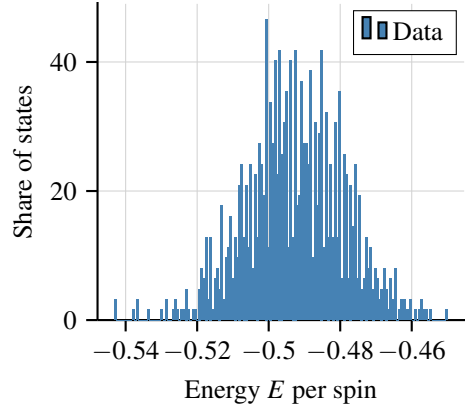


Figure 6: For $M = 10^6$

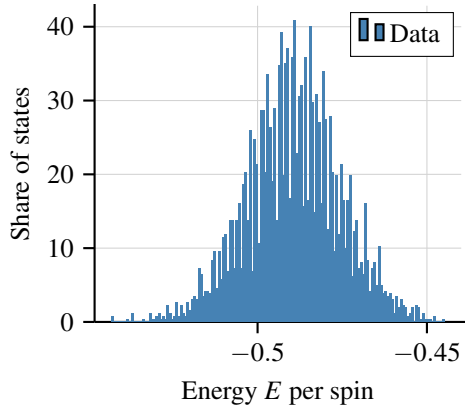


Figure 7: For $M = 4 \cdot 10^6$

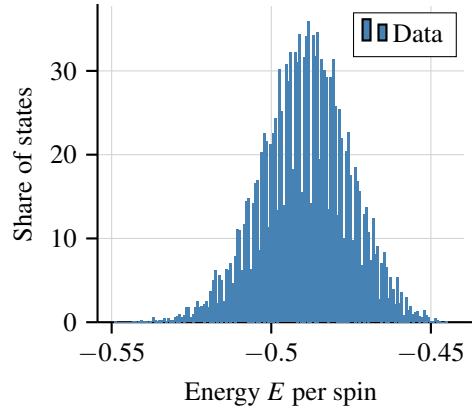


Figure 8: For $M = 10^7$

reached equilibrium. After that we varied the number of steps M of the simulation and plotted the distribution of energies. The results can be seen in figures 5 to 8. One sees that as M increases the distribution approaches the Maxwell-Boltzmann distribution.

Energies in dependence of the temperature

In this experiment we plotted the energies in dependence of the temperature for a system with parameters $q = 2$ and $q = 10$. Our aim was to see the critical temperature $T_c = 1/(\ln(1 + \sqrt{q}))$. For this we wanted to simulate a system of maximal size. Since a sampling size beyond 10^7 seemed unreasonable and the temperature would be in the region of $T \approx 1$ the previous experiment suggested a maximal system size of $L = 50$. For larger systems the distribution plot shown in figure 8 would become degenerate.

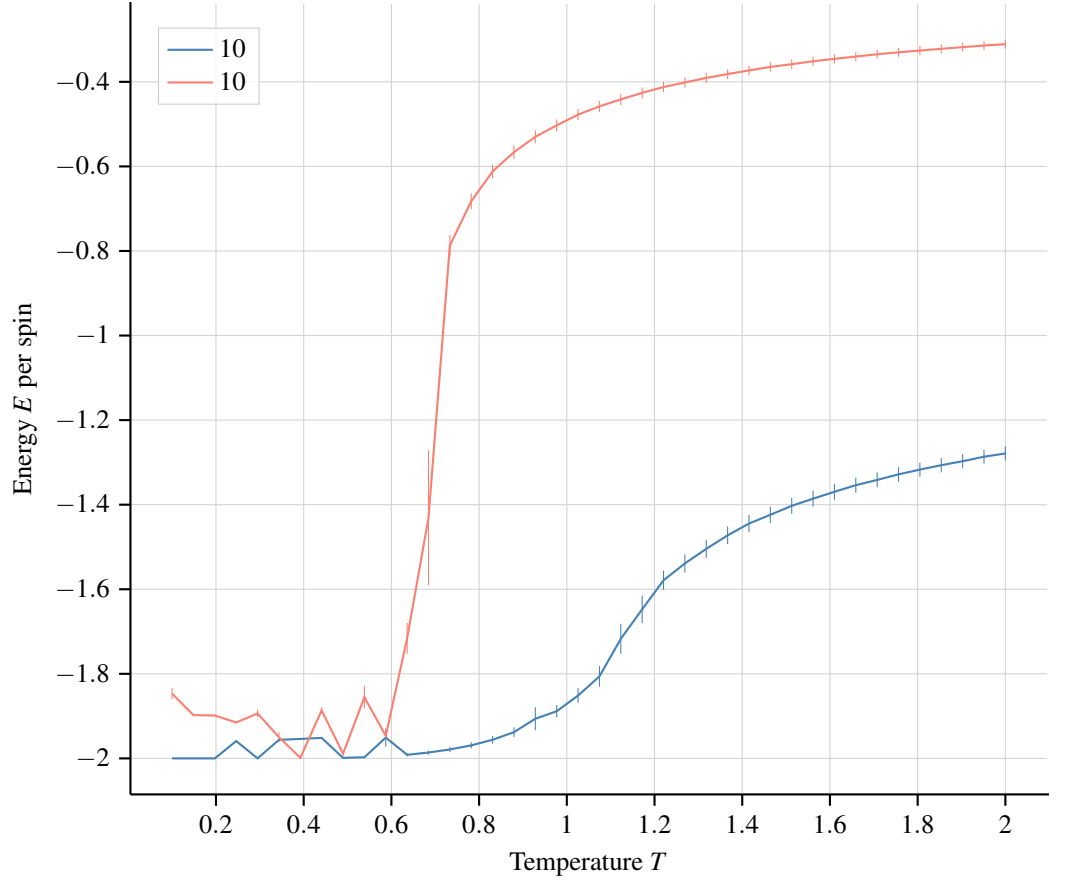


Figure 9

So we run the experiment with a sampling size of $M_{\text{sampling}}=10E7$ and chose the parameter n for the moving averages to be 10^6 . The results of the experiment are shown in plot. The error bars in the plot are the standard deviation of the energies.

We can see from figure 10 that it takes the simulation in most cases roughly $0.25 \cdot 10^7$ steps to reach an equilibrium. For cold temperatures with $q = 10$ this takes however far longer since the system starts in the 'hot' state.

In another series of experiments we would like to see how this plot depends on the system size L . The results can be seen in plots for system sizes $L = 10$ and $L = 30$.

Plotting the energy distribution around the critical temperature

In a final experiment we plotted the distribution of energies around the critical temperature for a grid size of $L = 50$ and parameter $q = 10$. The number of sampling steps was

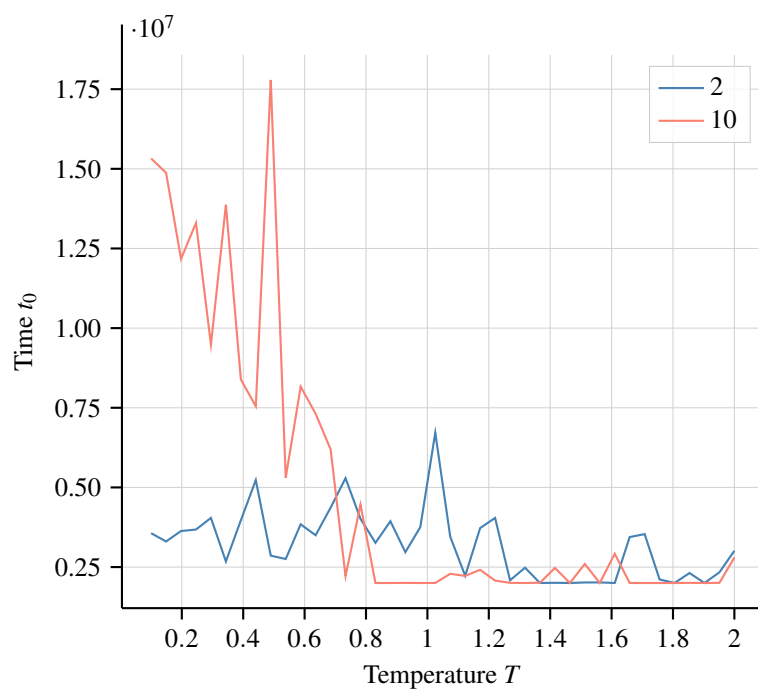


Figure 10: Time t_0 until the equilibrium is reached.

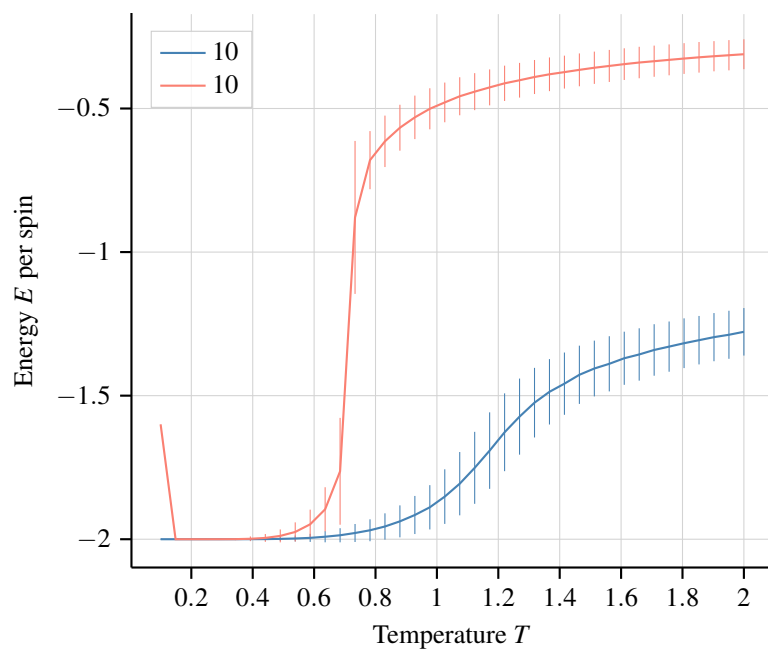


Figure 11: $L = 10$.

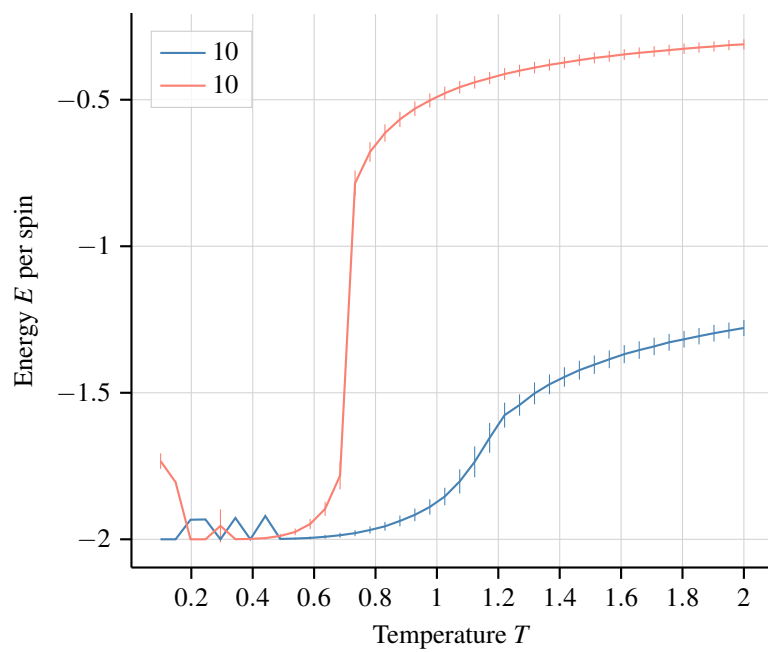


Figure 12: $L = 30$.

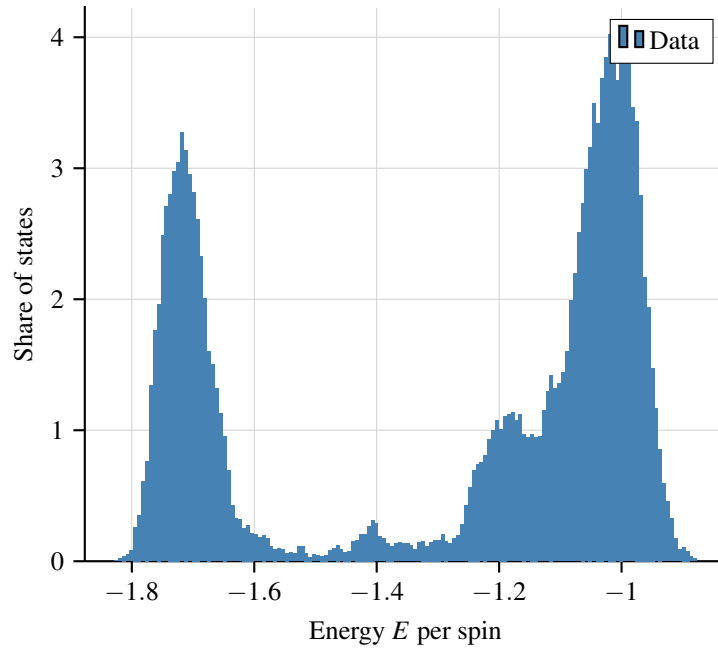


Figure 13: Distribution of energies for the temperature $T = 0.697$.

taken to be $M_{sampling} = 10^8$. In figure 13 we see that the distribution of energies splits up at the temperature $T = 0.697$. In figure 14 the distribution has shifted to the left for the slightly lower temperature $T = 0.696$. Correspondingly for the slightly higher temperature $T = 0.698$ we see in figure 15 that the distribution has shifted to the right.

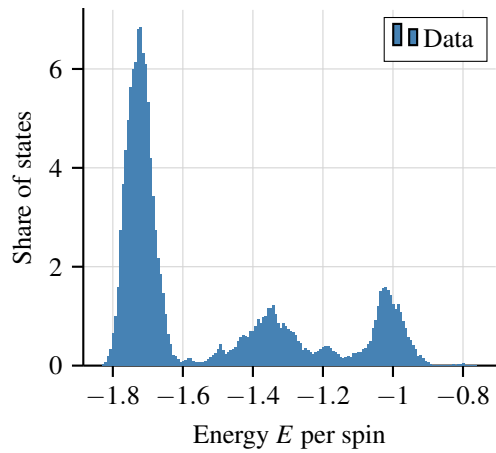


Figure 14: Distribution of energies for the temperature $T = 0.696$.

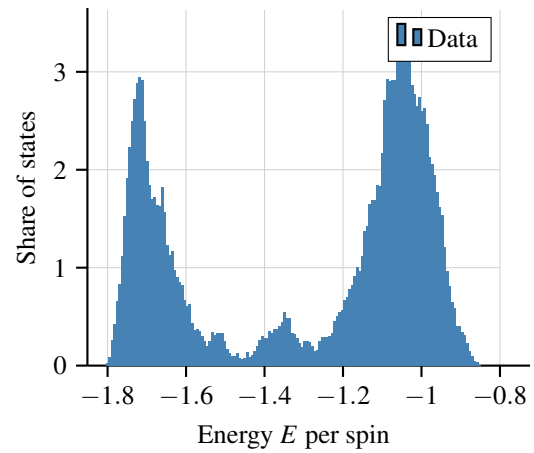


Figure 15: Distribution of energies for the temperature $T = 0.698$.

TODO:

- sanity test hot vs. cold start
- energy evolution for maxwell-distribution test
- phase transition test - ζ E-T plots and t_0 plot
- distribution based on energy
- spell-check

Bibliography

- [1] computational-science-HT23, *Github repository to the project*. Online, 2023. [Online]. Available: <https://github.com/TheoKoppenhoefer/computational-science-HT23>.