Report for the Course Modelling in Computational Science, HT23

Project 1: The Potts model

Theo Koppenhöfer

Lund

September 16, 2023

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. In the first part of this report we will describe the Potts model, the Monte-Carlo algorithms and some technicalities regarding the implementation. In the second part we will describe several experiments and the results they yielded. The report and the Python implementation can be found online under [1].

The setup

The Potts model

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

$$s\colon \{1,\dots,L\} \times \{1,\dots,L\} \to \{1,\dots,L\}\,.$$

One assigns this state an energy via

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

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 aim is to evaluate the integral

$$\langle E \rangle = \int E(s)p(s)\,\mathrm{d}s\,. \tag{1}$$

Here p is the density corresponding to the Boltzmann distribution, i.e.

$$p(s) = \exp(-E(s)/T)/Z$$

with Z a normalisation constant. To evaluate the integral in (1) we will use random sampling of E with the samples distributed according to the Boltzmann distribution. In other words, we will use a Monte-Carlo simulation.

The Monte-Carlo 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.

Input : Initial data s, L, T, q

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

Algorithm 1: Metropolis

Input : Initial data s, L, T, q

for $k = 0, 1, \ldots$ **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

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.

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

$$\text{ma}_2 \leq \text{ma}_1 \tag{2}$$

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 (2). 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.

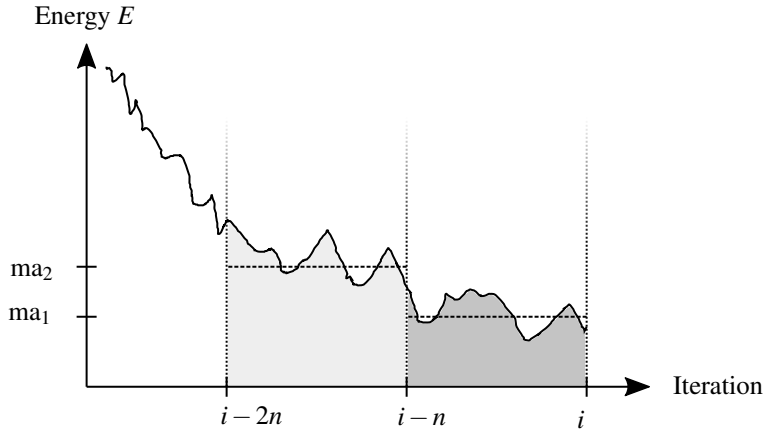

Figure 1: A visualisation of the moving averages.

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 unaestetic 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. Indeed, the last time I checked this was the case.

We also ran the simulation for both the Metropolis and the heat-bath algorithms and for a hot start and a cold start and compared the results. In the hot start the state s is randomly initialised and in the cold start s is initialised to have a constant value. We set the parameter $q = 2$ and the gridsize to $L = 100$. In a first experiment the temperature was set to $T = 100$. The result for the energies is plotted in figure 2. One can see that for both initialisations and for both algorithms the energy converges to a fixed value. Since the temperature is relatively 'hot' the hot start reaches equilibrium faster. In a second experiment the temperature was set to $T = 0.1$ and the result can be seen in figure 3. Here too the energies are starting to converge to a fixed lower value but for the hot starts

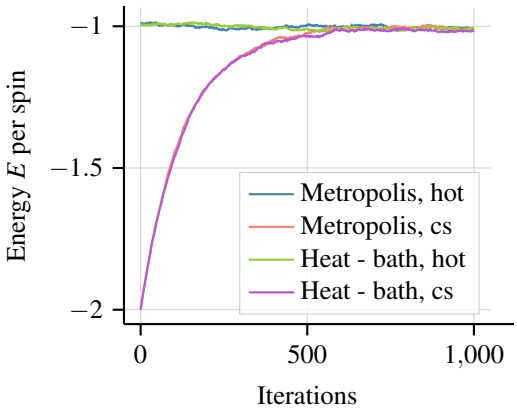

Figure 2: Energy evolution for the temperature $T = 100$

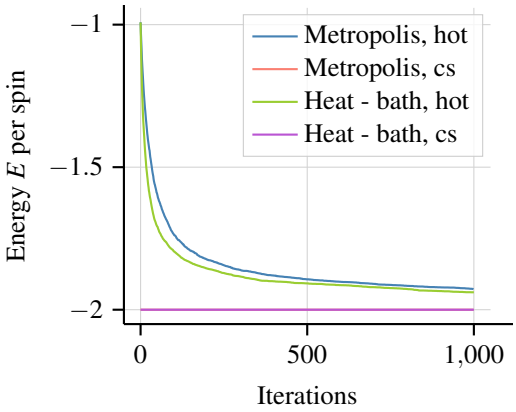

Figure 3: Energy evolution for the temperature $T = 0.1$

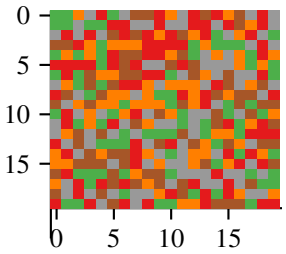

Figure 4: State for temperature $T = 0.1$.

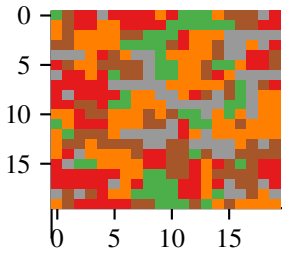

Figure 5: State for temperature $T = 1$.

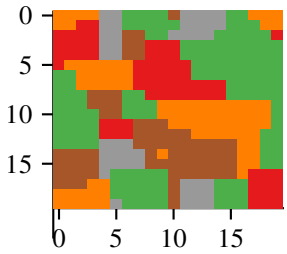

Figure 6: State for temperature $T = 100$.

this process takes far longer. As expected the energy the system tends to is far lower than for the test with higher temperatures.

State plots

We also plotted the state of the system for a system size $L = 20$ with parameter $q = 5$ after $M = 10^4$ steps for different temperatures. In figures 6 to 4 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 $L = 50$, the spin states to $q = 10$ and the temperature to $T = 1$. We then run the simulation until the system

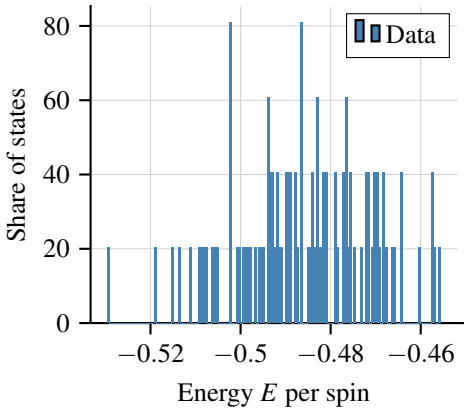

Figure 7: For $M = 10^5$

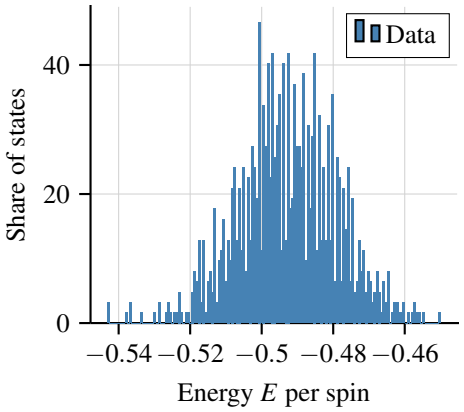

Figure 8: For $M = 10^6$

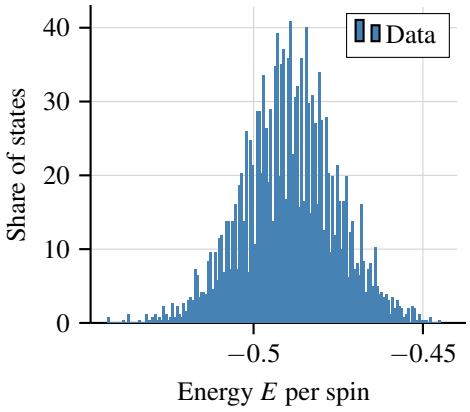

Figure 9: For $M = 4 \cdot 10^6$

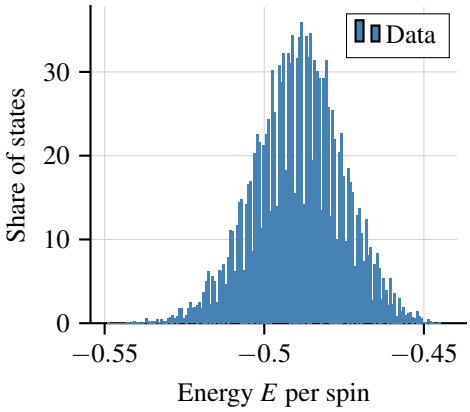

Figure 10: For $M = 10^7$

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 7 to 10. 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 10 would become degenerate.

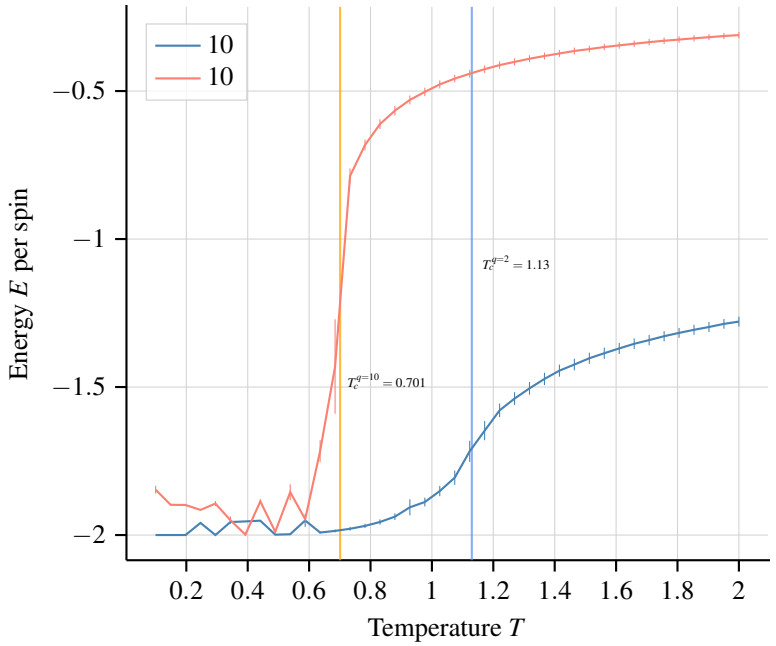

Figure 11

So we run the experiment with a sampling size of `M_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 12 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 $L = 30$. One can notice that as the state size L becomes decreases the standard deviation for the energies per spin increases. It is also noticable that the curve for the parameter $q = 10$ smoothens out.

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 taken to be $M_sampling = 10^8$. In figure 15 we see that the distribution of energies splits up at the temperature $T = 0.697$. In figure 16 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 17 that the distribution has shifted to the right.

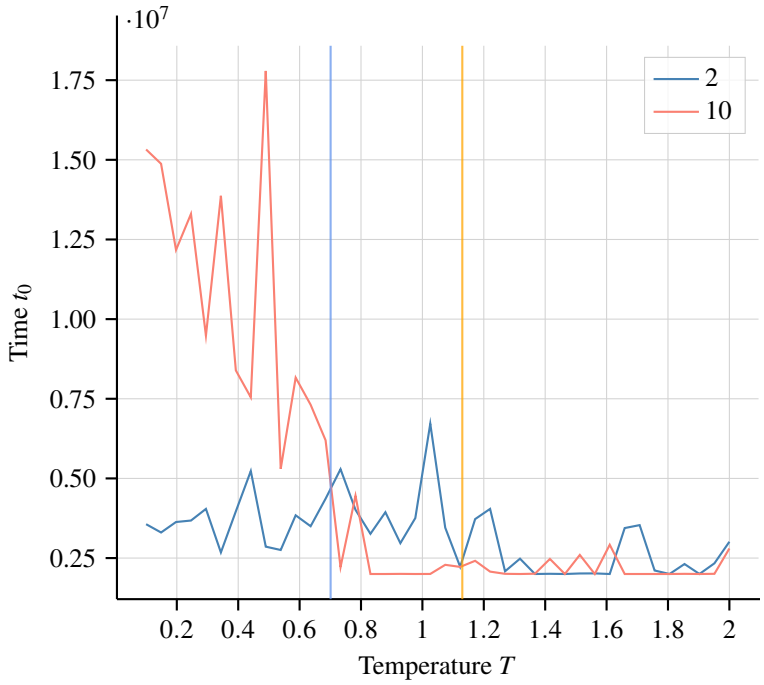

Figure 12: Time t_0 until the equilibrium is reached.

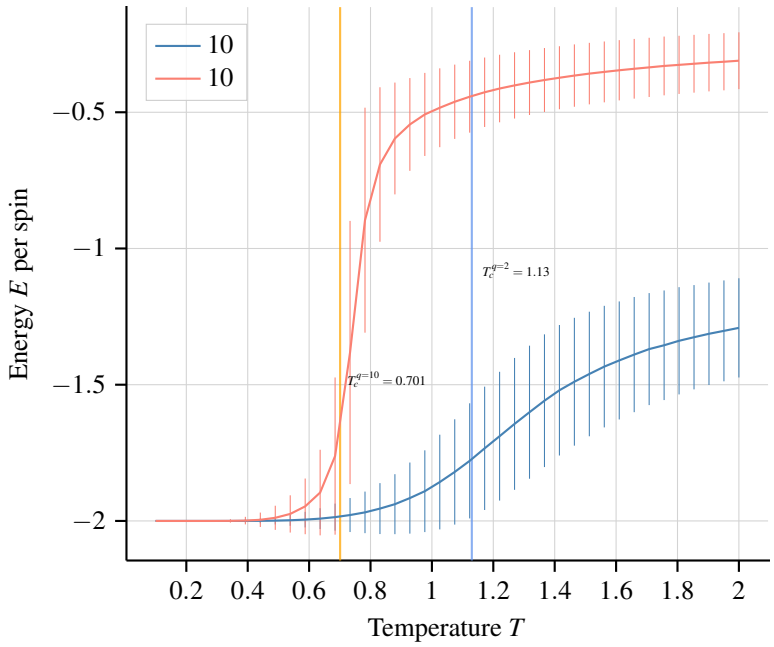
10
10
−0.5
−1
−1.5
−2
$T_c^{q=2} = 1.13$
$T_c^{q=10} = 0.701$
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
Energy E per spin
Temperature T

Figure 13: $L = 5$.

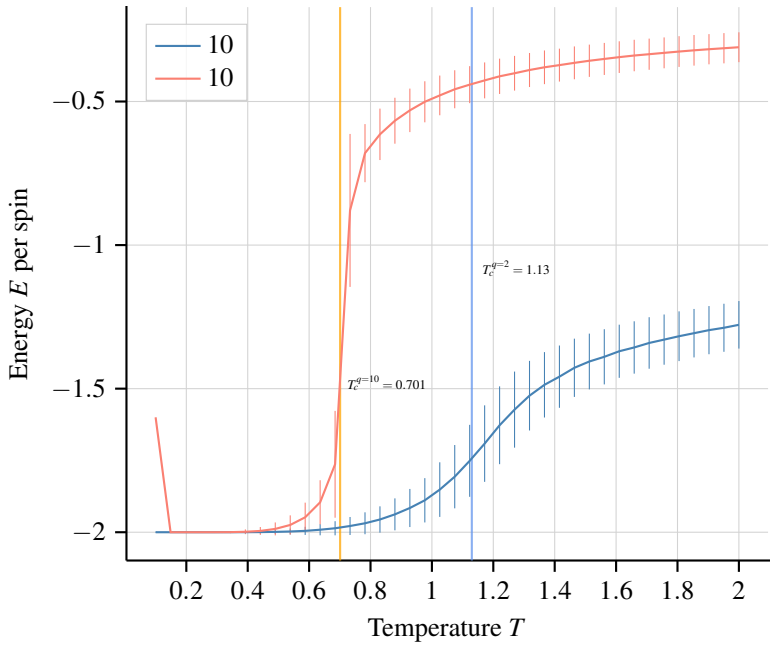

Figure 14: $L = 10$.

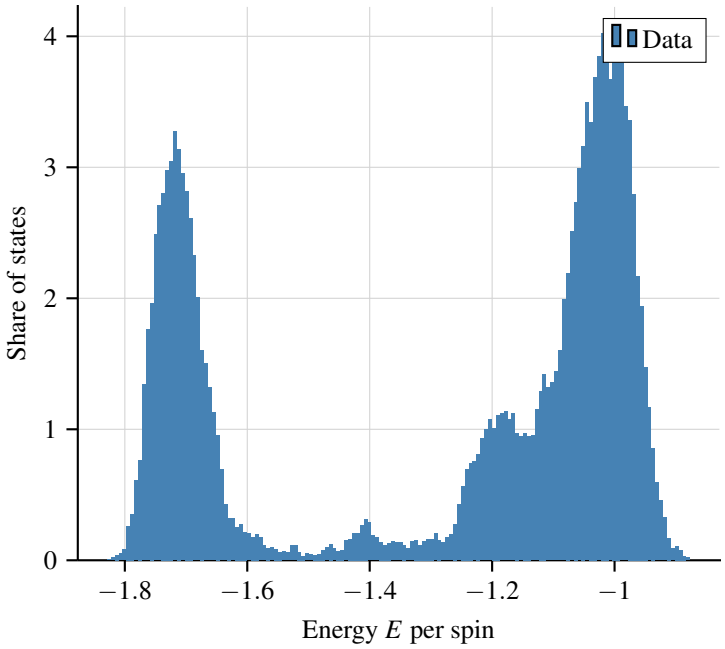

Figure 15: Distribution of energies for the temperature $T = 0.697$.

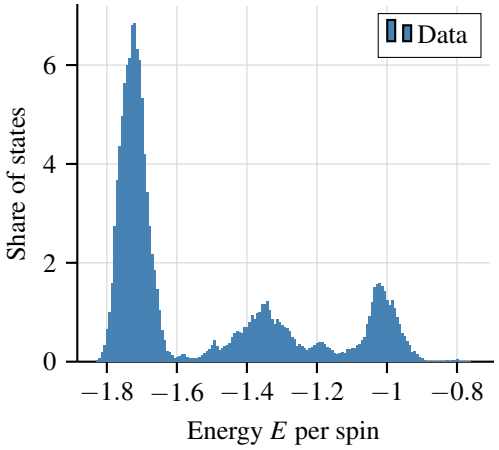

Figure 16: Distribution of energies for the temperature $T = 0.696$.

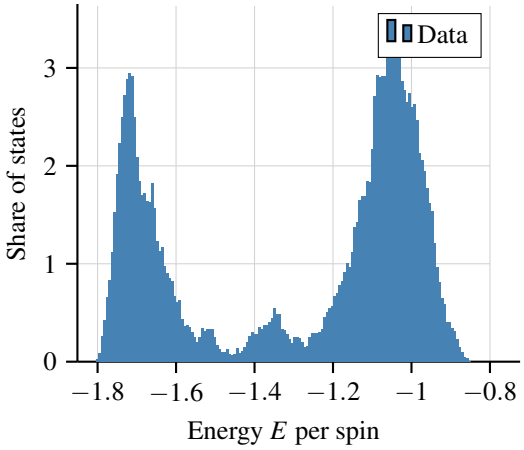

Figure 17: Distribution of energies for the temperature $T = 0.698$.

TODO:

- distribution based on energy
- explain why it approaches the maxwell-boltzmann distribution
- proof-read
- spell-check

Bibliography

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