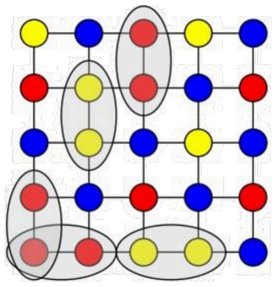
OPEN ACCESS



Simulation of 2D 3-State Potts Model

A Metroplis Monte Carlo Approach

Xiaoxuan Yu[1]

[1]College of Chemistry and Molecular Engineering, Peking University

Abstract

The Potts model is a generalization of the Ising model, which is a model of ferromagnetism in statistical mechanics. By using the Metropolis algorithm, a 2D 3-state Potts model is simulated under different temperatures. I studied the internal energy, magnetization, specific heat and magnetic with reference to the temperature and found that the system undergoes a phase transition at a critical temperature around 0.94.

Keywords Monte Carlo, Metropolis Algorithm, Potts Model, Phase Transition

COURSE PROJECT

Submitted Nov 27, 2023
Due Date Nov 30, 2023

Correspondence to
Xiaoxuan Yu
xiaoxuan_yu@pku.edu.cn

Data Availability
Associated code repository is available on GitHub.

Honesty Statement
The author declares that the project was completed independently and without plagiarism.

1. BACKGROUND

1.a. *Spin models*

The Potts model is one of a group of models called spin models. In a spin model, each site on a lattice is assigned a spin variable. The spin variable can take on a discrete set of values, which are usually integers.

1.a.i. *Ising model:*

The Ising model is the most basic spin model. In the Ising model, the spin variable can take on only two values, +1 or −1. The Ising model is a model of ferromagnetism in statistical mechanics. The Hamiltonian of the Ising model is given by

$$H = -J \sum_{\langle i,j \rangle} S_i S_j \tag{1}$$

where S_i is the spin variable at site i, J is the coupling constant, and the sum is over all pairs of nearest-neighbor sites.

1.a.ii. *Potts model:*

The Potts model is a generalization of the Ising model. In the Potts model, the spin variable can take on q different values, where q is an integer greater than or equal to 2. The Hamiltonian of the Potts model is given by

$$H = -J \sum_{\langle i,j \rangle} \delta(S_i, S_j) \tag{2}$$

where S_i is the spin variable at site i, J is the coupling constant, and the sum is over all pairs of nearest-neighbor sites. The Kronecker delta function $\delta(S_i, S_j)$ is defined as

$$\delta(S_i, S_j) = \begin{cases} 1 \text{ if } S_i = S_j \\ 0 \text{ if } S_i \neq S_j \end{cases} \tag{3}$$

If magenetic field is applied, an extra term is added to the Hamiltonian

$$H = -J \sum_{\langle i,j \rangle} \delta(S_i, S_j) - h \sum_i S_i \tag{4}$$

where h is the magnetic field strength.

1.b. Interested physical quantities in Potts model

Pratically, physicists are interested in the following physical quantities in Potts model:

1.b.i. *Internal energy:*

$$u = \frac{\langle H \rangle}{N^2} \tag{5}$$

1.b.ii. *Specific heat:*

$$c = \frac{k_B \beta^2 \left(\langle H^2 \rangle - \langle H \rangle^2 \right)}{N^2} \tag{6}$$

where k_B is the Boltzmann constant and $\beta = \frac{1}{k_B T}$ is the inverse temperature.
1.b.iii. *Magnetization:*

$$m = \frac{\left\langle \sum_i S_i \right\rangle}{N^2} \tag{7}$$

1.b.iv. *Characteristic length:* Define the spatial correlation function

$$C(i,j) = \langle S_i S_j \rangle - \langle S_i \rangle \langle S_j \rangle \tag{8}$$

Then we can define

$$\Gamma(k) = C(i,j)\big|_{|i-j|=k} \approx \frac{1}{4N^2} \sum_i \sum_{j \in S_i} C(i,j) \tag{9}$$

where

$$S_i = \{ i | i - j = \pm(k, 0) \text{ or } (0, k) \} \tag{10}$$

And the characteristic length ξ is the length that $\Gamma(k)$ decays to 0. Thus the correlation length is given by

$$\Gamma(k) \propto \Gamma_0 \exp(-k/\xi), \quad k \gg 1 \tag{11}$$

1.c. *Phase transition in Potts model*

The Potts model undergoes a phase transition at a critical temperature T_c. Below T_c, the system is in a ferromagnetic phase, where the spins are aligned. Above T_c, the system is in a paramagnetic phase, where the spins are randomly oriented. The critical temperature T_c depends on the coupling constant J and the dimensionality of the system. For a 2D square lattice, the critical temperature is given by

$$T_c = \frac{J}{k_B \ln(1 + \sqrt{q})} \tag{12}$$

where k_B is the Boltzmann constant and q is the number of spin states. In the simulation approach, the critical temperature can be determined by finding a peak in the specific heat.

2. Problem setup

In the simulation, a 2D square lattice with periodic boundary is used as the target system. The lattice has size $N = 32$. For simplicity, we set the coupling constant $J = 1$ and the magnetic field strength $h = 0$. Besides we use the reduced temperature s.t. $k_B = 1$. q is set to 3, which means the spin variable can take on 3 different values, 1, 2, and 3.

The initial state of the system is randomly generated. The simulation is performed under different reduced temperatures T from 0.5 to 2.5. For each temperature, the system is evolved for 500,0000 steps. The trajectory of the system is recorded every 100 steps and thus the physical quantities are calculated with the same interval.

We selected the last half of the trajectory for analysis. The first half of the trajectory is used to ensure the system has reached equilibrium. The physical quantities are calculated by averaging over the last half of the trajectory.

3. Implementation of the simulation

3.a. *Setup the dependencies*

In the simulation, `numpy` is used for the main computation. For better performance, `numba` is used to compile the functions just in time. `multiprocessing` is used to parallelize the computation, and `functools.partial` is used to create partial functions for the parallelization.

```
import numpy as np
from functools import partial
from numba import njit, prange, objmode, jit
import multiprocessing as mp
```

3.b. *The Potts Hamiltonian*

The Hamiltonian of Potts model is implemented as the following `potts_energy` function

```python
@njit
def potts_energy(spins, J, h):
    # 获取形状
    N = spins.shape[0]
    # 计算相邻自旋互作用能
    E_J = J * np.sum(spins[:-1, :] == spins[1:, :])
    E_J += J * np.sum(spins[:, :-1] == spins[:, 1:])
    # 计算外磁场能
    E_J += h * np.sum(spins)
    # 求和得到能量
    E = -E_J
    return E
```

The implementation is straightforward. Using the slice operation, we can calculate the energy of the system by comparing the neighboring spins vertically and horizontally, and then summing up the results. The influence of the magnetic field is calculated by naively summing up all the spins. An decorator `@njit` is added to the function to compile it just in time. This will significantly improve the performance of the function.

3.c. *The Metropolis algorithm*

By splitting the Metropolis algorithm into the main loop and the operation of a single step, it is much more easier for `numba` to compile the code. The single step operation is implemented as the following `potts_mc_step` function

```python
@njit(fastmath=True)
def potts_mc_step(spins, J, h, beta, q):
    # 获取形状
    N = spins.shape[0]
    # 选取随机格点
    i = np.random.randint(0, N)
    j = np.random.randint(0, N)
    # 建议新的自旋状态
    spins_new = spins.copy()
    spins_new[i, j] = np.random.randint(1, q + 1)
    # 直接计算能量差，由于只改变了一个格点，只需要计算该格点的能量差
    Eij_old = 0
    Eij_new = 0
    '''
    Some code are omitted here for simplicity
    '''
    dE = Eij_new - Eij_old
    # 判断是否接受
    if dE <= 0 or np.random.rand() < np.exp(-beta * dE):
        spins = spins_new
    return spins
```

It is a standard implementation of the Metropolis algorithm. The only thing to note is that the energy difference `dE` is not calculated by comparing the energy of the new state and the old state. Instead, we only calculate the energy difference of the changed spin. The time complexity is thus reduced to $O(1)$, which is much better than the naive implementation with time complexity $O(N^2)$. The main loop of the Metropolis algorithm is implemented naively and is not shown here. The sampled trajectories are recorded in the format of a `numpy` array as `.npy` files in the disk for further analysis.

3.d. Parallel simulation of different temperatures

The simulation of different temperatures is parallelized by using the `multiprocessing` module. The main idea is to create a partial function of the Metropolis algorithm with a fixed temperature. Then we can use the `map` function of the `multiprocessing.Pool` class to run the simulation of different temperatures in parallel. The implementation is shown as the following `potts_simulate_parallel` function

```python
def potts_simulate_parallel(init_spin, J, h, q, k, T_series, n_steps, 
n_step_save=100):
    # 创建进程池
    pool = mt.Pool(8)
    # 定义部分函数
    potts_mc_partial = partial(
        potts_mc_mt,
        spins=init_spin,
        J=J,
        h=h,
        q=q,
        n_steps=n_steps,
        n_step_save=n_step_save,
    )
    # 运行并行模拟
    pool.map(potts_mc_partial, T_series)
    return 0
```

The `potts_mc_partial` function is a partial function of the `potts_mc_mt` function, which is the main loop of the Metropolis algorithm, in which parameters other than temperature are all fixed. The `potts_mc_partial` function is then mapped to the `T_series` array, which contains the temperatures of the simulation. The `potts_mc_partial` function is run in parallel with 8 processes by using the `multiprocessing.Pool` class. The parallelization of h can be implemented in a similar way.

3.e. Data analysis

The interested physical quantities are calculated by averaging over the last half of the trajectory. The implementation is shown is the attached notebook `Post-simulation.ipynb`. Simply speaking, the data is loaded from the disk and then averaged over the last half of the trajectory. For internal energy, specific heat, and magnetization, it is done naively. For the characteristic length, the spatial correlation function is calculated first and then the characteristic length is calculated by fitting the correlation function using `scipy.optimize.fitting_curve` with the exponential function.

4. Numerical results

4.a. Internal energy and specific heat