Report 4: Metropolis Monte Carlo sampling of Ising model spin configurations in 1D and 2D

Don Winter[1] **and Madhulan Suresh**[2]

[1]don.winter@rwth-aachen.de, Mat. No. 431380
[2]madhulan.suresh@rwth-aachen.de, Mat. No. 429481

Introduction

In this exercise, we explore several properties of the Ising model in both one and two dimensions using numerical simulation. Due to the large computational complexity of this combinatorial task, we employ the Metropolis algorithm in a Monte Carlo loop in order to achieve importance sampling of the configuration space. We thereby are able to compute expectation values of internal energy U, specific heat C and average magnetization M per spin, which we afterwards compare to the analytical solutions found by Ising and Onsager in 1925 and 1944, respectively. Finally, we take a closer look at the phase transition of magnetization in the two-dimensional case and give an interpretation of our results.

Simulation model and method

As the task is to numerically obtain expectation values for properties of the Ising model, we will first introduce the theoretical model together with its analytical solution and afterwards the numerical method used for solving it.

Ising model

The Ising model is a mathematical model in statistical mechanics which allows to study phase transitions in ferromagnetic materials. It describes spins of value ± 1 distributed on a lattice in which only nearest neighbors are allowed to interact via a coupling strength J. Furthermore, the original model includes a controllable external magnetic field of strength h at each site. For this exercise we will use $J = 1$ and $h = 0$, and therefore obtain for the energy of the system:

$$E(S) = -\sum_{\langle ij \rangle} S_i S_j = \begin{cases} -\sum_i^{N-1} S_i S_{i+1} & \text{one-dimensional case} \\ -\sum_i^{N-1} \sum_j^N S_{i,j} S_{i+1,j} - \sum_i^N \sum_j^{N-1} S_{i,j} S_{i,j+1} & \text{two-dimensional case,} \end{cases} \tag{1}$$

for a spin $S_i \in \{-1, 1\}$ over all N ($N \times N$) lattice sites $\{S_i\}$ ($\{S_{i,j}\}$) on a one(two)-dimensional lattice with free boundary conditions. From this energy function we can further calculate the analytical solution of the partition function Z for canonical ensembles. For the one-dimensional case we obtain:

$$Z(\beta) = \sum_{\{S_i\}} e^{-\beta E(S_i)} = 2(2cosh\beta)^{N-1}, \tag{2}$$

for inverse temperature $\beta = (k_B T)^{-1}$, which we choose in units of $k_B = 1$. With the partition function as normalization for the configuration probabilities $P(S_i)$ we can further calculate the expectation value of any observable $\langle O \rangle$ on the system as

$$\langle O \rangle = \sum_{\{S_i\}} P(S_i) O(S_i) = \frac{1}{Z(\beta)} \sum_{\{S_i\}} O(S_i) e^{-\beta E(S_i)}, \tag{3}$$

from which we can derive the analytical solution for $U/N = \langle E \rangle / N = -\frac{N-1}{N} tanh\beta$, and thus $C/N = -\frac{\beta^2}{N} \frac{\partial U}{\partial \beta} = \frac{N-1}{N} (\beta / cosh\beta)^2$. For the two-dimensional case, the analytical solution is a lot more involved and will therefore not be discussed here. The analytical expression for average magnetization per spin, i.e. the expectation value of magnetization M/N^2 can be obtained analogous to Eq. 3, which analytically evaluates to $M/N^2 = (1 - sinh^{-4}(2\beta))^{1/8}$ if $T < T_C$ and to $M/N^2 = 0$ otherwise, where $T_C = \frac{2}{ln(1+\sqrt{2})}$ is the critical temperature at which a phase change occurs.

Monte Carlo simulation

From Eq. 3, we can already see that the exact solution of expectation values can rapidly become hard to solve with growing N. Indeed, the task of calculating $Z(\beta)$, which only needs to be done once, already becomes an impossible task for larger N. If we want to calculate the expectation values anyway we need to trade off accuracy of an exact solution with computational complexity by randomly sampling from all possible spin configurations. Thereby, we can obtain an approximation of the observable's expectation value by averaging over the expectation values for our sampled configurations. The result thus grows in accuracy as we increase the number of samples. The repeated (uniform) drawing of random samples from a configuration space is known as Monte Carlo simulation. When all configurations in our sampling space are equally important for the calculation of an expectation value, then Monte Carlo sampling is a good approach. However, it is more often the case that only a few configurations provide important contributions and thus in order to approach the expectation we would need a lot of samples. In this case we want to further trade-off accuracy for complexity and sample only in the subspace of "important" configurations. This is called *Importance* sampling and the Metropolis algorithm is one simple method to produce such samples.

Metropolis algorithm

The Metropolis algorithm iteratively explores the configuration space and produces one sample which is not uniformly drawn but selected due to some (posterior) probability distribution over the configuration space. In the case of the Ising model, we are interested in finding spin configurations which contribute most to the system energy at a given temperature. As we know from statistical mechanics, these energies are Boltzmann-distributed. Thus, we utilize the distribution to guide the selection process of our samples. The algorithm tries N ($N \times N$) random spin flips and chooses one of these probabilistically by considering the likelihood of each proposal configuration's energy in terms of the Boltzmann distribution. By iteration of the algorithm, we obtain thereby configurations sampled from the same (posterior) distribution over the configuration space. One Metropolis run can be implemented as

1. Choose a random initial spin configuration $S = \{S_i\}$ where $S_i \in \{-1, 1\}$. ("Hot start")
2. Randomly flip one spin S_i in S: $S_i \to -S_i$. This yields a new proposal S'.
3. Calculate the energy difference between the configurations:

$$\Delta E = E(S') - E(S). \tag{4}$$

4. Accept S' as new configuration with probability

$$P(S \to S') = \begin{cases} 1 & \text{if } \Delta E < 0 \\ e^{-\beta \Delta E} & \text{if } \Delta E \geq 0. \end{cases} \tag{5}$$

5. Repeat steps 2 to 4 N ($N \times N$) times for one(two)-dimensional case.

At the end of the algorithm we obtain one configuration S', randomly selected according to a probability distribution over configuration space which weights each configuration according to its energy with respect to the probability of energies in the Boltzmann distribution for a fixed T. The sampled configuration represents one sample of our Monte Carlo simulation, in which we can compute the average values of energy, specific heat and magnetization per spin.

Furthermore, starting from a complete random configuration we must iteratively run the Metropolis algorithm for some time before we can start using the samples to calculate our desired statistics. This process is called thermalization and it is dependent on both the temperature and the size of the configuration space. In our simulation we account for thermalization by running the algorithm $N_{therm} = N_{samples}/10$ times. Besides thermalization the samples which the algorithm generate are clearly dependent of each other (because after all we use a previous sample configuration as starting point to produce the next sample). Thus samples generated at one time step can correlate with samples generated at another time, which is not good as one of the assumptions for Monte Carlo sampling is that the samples are independent. This is referred to as autocorrelation and for the work of this report we want to stress that we did not take any precautions against autocorrelation. However, at the relatively small number of spins we used in our simulations one can argue weather autocorrelation has any effect at all.

Specifics about our implementation

Although our implementation follows closely the above discribed algorithm, there were several possible options on how to implement the calculation of transition probabilities and energy differences. For the one-dimensional case we noticed that the difference between two configuration energies can be efficiently computed by only considering the neighborhood of a proposed spin flip location. Instead of calculating naively the energy difference $\Delta E = E(S') - E(S)$, we choose to instead calculate $\Delta E = 2S_i(S_{i+1} + S_{i-1})$, for which $S_{i\pm 1} = 0$ at the boundaries since we have free boundary conditions. This allows us to compute $P(S \rightarrow S') = e^{-\beta \Delta E}$ efficiently inside the Metropolis algorithm.

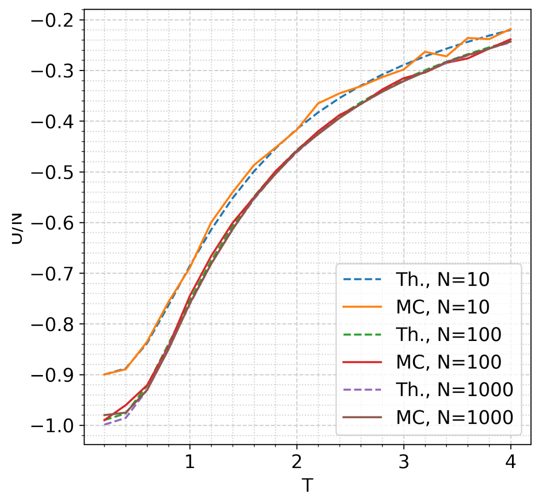
For the two-dimensional case we further noticed that the transition probabilities can actually be computed only once per Monte Carlo step, as they only depend on the temperature T for all possible cases which can occur when we randomly select a spin $S_{i,j} = \{-1, +1\}$ in the lattice which has a neighborhood of four other spins whose sum can only take the discrete values $\{-4, -2, 0, 2, 4\}$. Thus, by computing the 2×5 probability matrix before running the Monte Carlo loop, we were able to avoid calculation of this value anew within $N_{samples} \times N \times N$ steps inside the Metropolis algorithm. We computed the difference in energy between two configuration by $\Delta E = 2S_{i,j} \sum_{k,l} S_{k,l}$ where k, l are the nearest neighbor indices to i, j. Thus, the transition probabilities (of accepting a spin flip) are calculated by $P(S \rightarrow S') = e^{-\beta \Delta E}$, where $k_B = 1$.

Simulation results

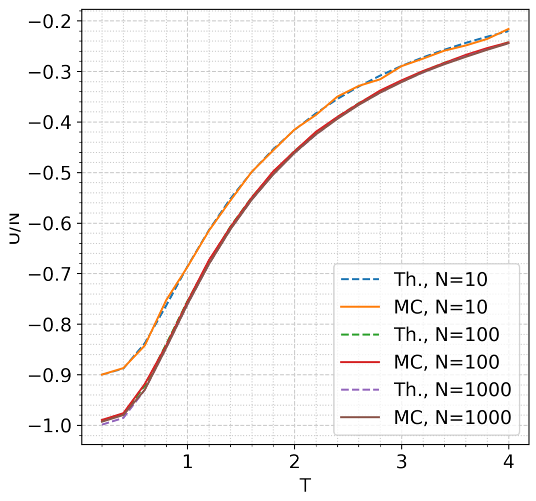
In the following we will first present our simulation results for the one-dimensional Ising model and subsequently for the two-dimensional case.

One-dimensional Ising model

As mentioned in the previous section, we generated samples from the Metropolis algorithm for which we tracked the energy per sample in a Monte Carlo loop for $N_{samples}$ steps (after $N_{samples}/10$ steps of thermalization for which we didn't track) per temperature value T. This procedure was repeated for $T = 0.2, 0.4, ...4.0$. For each T we then calculated the average energy $U/N = \langle E \rangle / N$ and the specific heat $C/N = (\langle E^2 \rangle - \langle E \rangle^2) T^{-2}/N = Var[E] T^{-2}/N$ per spin, where $k_B = 1$. The results can be seen in Fig. 1 and Fig. 2, respectively.



(a) 1000 samples per T



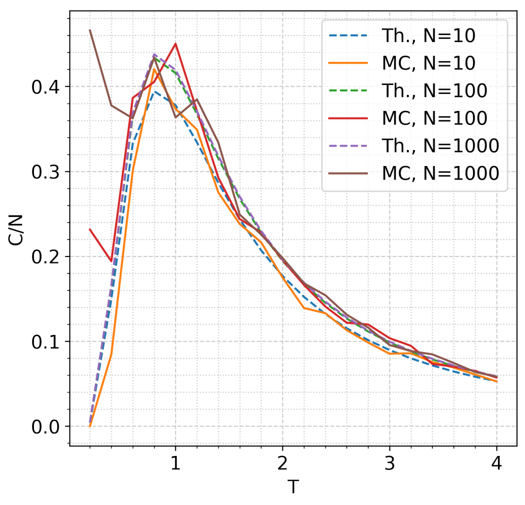
(b) 10000 samples per T

Figure 1. Average energy per spin as a function of temperature T for one-dimensional spin systems of N spins. The dashed lines represent exact analytical solutions and the full lines the numerical values obtained from Monte Carlo sampling.

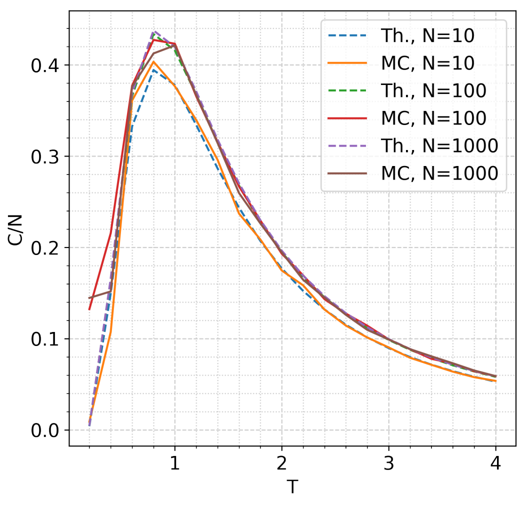
We see that especially the small spin system ($N = 10$) deviates from the theoretical solution in the low temperature region. For this system, we also see that the numerical result for 1000 samples (Fig. 1a) fluctuates more than for larger systems. Futhermore, although we increased the number of samples from Fig. 2a to Fig. 2b by one order of magnitude, the observable estimates of this system barely improve. For the other sytems ($N = 100, 1000$) on the other hand we see a clear improvement of the simulation results with increasing sample size. This is especially visible in the low temperature regions.

Two-dimensional Ising model

For $N_{samples} = 1000$ we see that in particular specific heat and magnetization are either far off or fluctuate wildly for larger spin systems of $N = 50, 100$ below $T_C = 2/ln(1 + \sqrt{2}) \approx 2.27$. We would expect to see a peak at $T = T_C$, where the phase transition



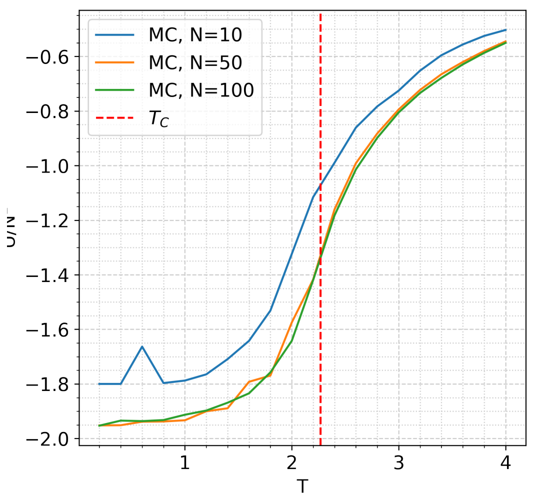
(a) 1000 samples per T



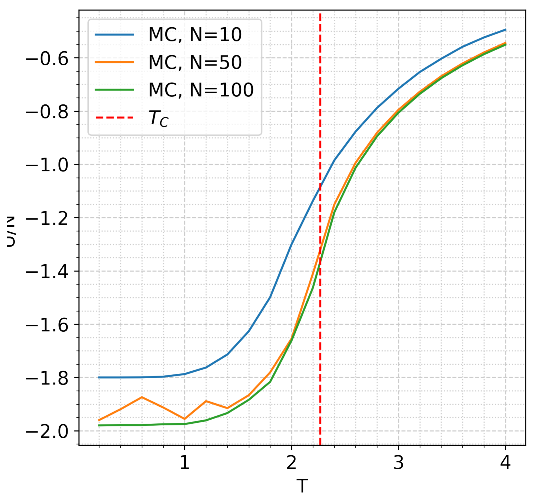
(b) 10000 samples per T

Figure 2. Specific heat per spin as a function of temperature T for one-dimensional spin systems of N spins. The dashed lines represent exact analytical solutions and the full lines the numerical values obtained from Monte Carlo sampling.

occurs, as we see from Fig. 3 that we have the largest change in U with temperature at this point. The fluctuations in C/N^2



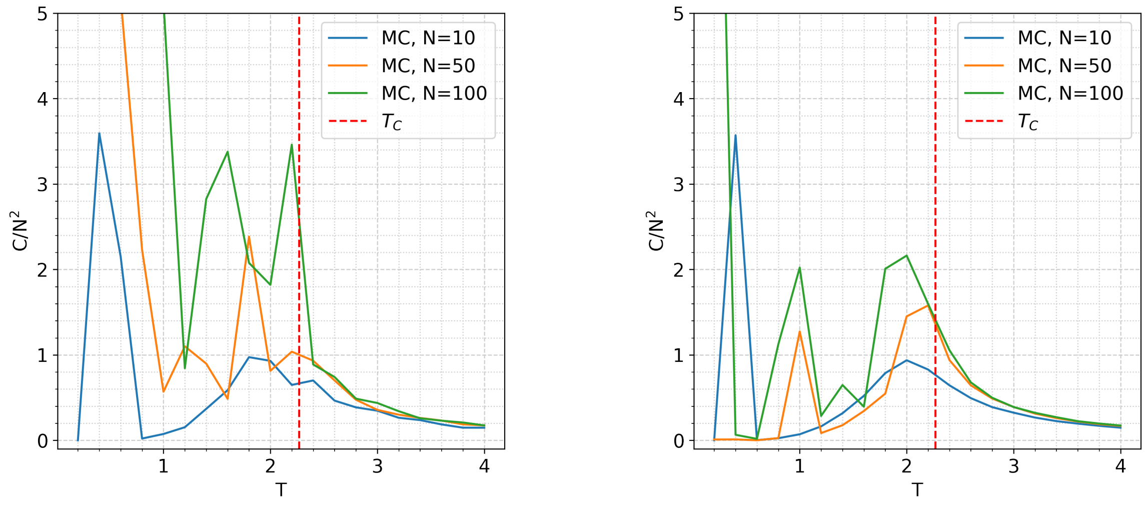
(a) 1000 samples per T



(b) 10000 samples per T

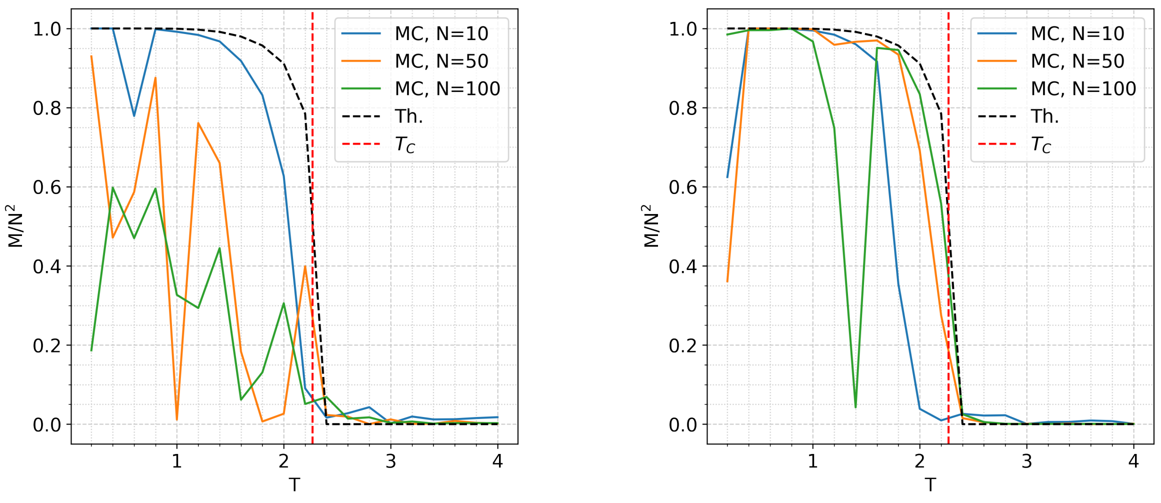
Figure 3. Average energy per spin as a function of temperature T for two-dimensional spin systems of $N \times N$ spins. The full lines the numerical values obtained from Monte Carlo sampling and the red dashed line represents the critical temperature T_C.

therefore originate from (barely visible) changes in U/N^2 in the sub-T_C-region. For the small spin system ($N = 10$) we observe something weird – at temperatures below T_C all estimates seem fairly accurate, whereas for temperaturs above T_C especially the magnetization starts deviating starkly from the theoretical result. The situation for this system does not improve much when we increase the sample size to 10000 samples. For the larger systems on the other hand we can clearly see that the numerical result approaches the theoretical one for larger sample size. Only the specific heat for the system of $N = 100$ is still off in the



(a) 1000 samples per T **(b)** 10000 samples per T

Figure 4. Specific heat per spin as a function of temperature T for two-dimensional spin systems of $N \times N$ spins. The full lines represent the numerical values obtained from Monte Carlo sampling and the red dashed line represents the critical temperature T_C.



(a) 1000 samples per T **(b)** 10000 samples per T

Figure 5. Average magnetization per spin as a function of temperature T for two-dimensional spin systems of $N \times N$ spins. The black dashed line represents the exact analytical solution and the full lines represent the numerical values obtained from Monte Carlo sampling. The red dashed line represents the critical temperature T_C.

sub-T_C region. If we would run the Monte Carlo loop with increased $N_{samples}$ we would probably see an improvement in this estimate. The resulting measurements of average energy per spin U/N^2, specific heat per spin C/N^2 and absolute average magnetization per spin $|M|/N^2$ are shown in Figs. 3–5. We can see that the measurements of all these observables become more accurate with increased sample size. As mentioned already, we can see the phase transition where the spins relatively fast change from perfect alignment (all spins in either -1 or +1) to complete random alignment (50% +1, 50% -1). The direct proof of this is the magnetization in Fig. 5, whereas also the systems energy rapidly decreases to lower-energetic unordered states. The abruptness of the energy change in turn can be seen by the specific heat, which peaks at the critical temperature from which it exponentially decays to zero in both directions.

Discussion

Overall, we can see in our simulations that for large N or large T (above critical temperature T_C) our results for the one-dimensional Ising model nicely match the theoretically predicted values. As expected does the accuracy of the numerical results improve with sample size. In this case, we observe that especially the system of only 10 spins at temperatures below T_C still shows large error with respect to the analytical solution. This could be a result of the phanomenon of critical fluctuations, which are even more amplified in small spin systems, as one accepted spin flips has a larger impact on the average energy than for larger systems. For the two-dimensional Ising model we see more fluctuation and no correct estimations especially for small sample size $N_{samples} = 1000$. For $N_{samples} = 10000$, we observe an improvement in the estimates though it seems that the largest system of $N = 100$ needs even more samples to converge to the correct values.

Appendix

Common library imports

```python
import numba
import random
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
font = {'size': 14}
matplotlib.rc('font', **font)
random.seed(1234) # python random RNG (used in jit-functions)
np.random.seed(1234) # numpy RNG
```

One-dimensional Ising model

```python
@numba.jit(nopython=True)
def energy(S,N):
    '''Calculate energy of a configuration'''
    E = 0
    for i in range(1,N): # sum of neighbor products
        E -= S[i-1]*S[i]
    return E

@numba.jit(nopython=True)
def energy_diff(S,N,i):
    '''Calculate energy difference of current config. S to proposed spin flip at index i'''
    l = 0 if i-1 < 0 else S[i-1]
    r = 0 if i+1 > N-1 else S[i+1]
    return 2*S[i]*(r+l) # l_pair_old+r_pair_old - (l_pair_new+r_pair_new), *_pair_new = -*_pair_old

@numba.jit(nopython=True)
def metropolis_step(S,N,T):
    '''Metropolis algorithm'''
    for _ in range(N):
        i = random.randrange(N) # must use randrange, numba can't handle np.random.choice
        dE = energy_diff(S,N,i) # calculate energy difference
        if random.random() < np.exp(-dE/T): # accept/reject
            S[i] = -S[i]
    return S

@numba.jit(nopython=True)
def monte_carlo(S,T,N,N_samples,N_therm):
    '''Repeatedly call Metropolis-Hastings and calculate observables from samples'''
    for _ in range(N_therm):
        S = metropolis_step(S,N,T)
    E = np.zeros(N_samples) # resulting configuration energies
    for i in range(N_samples):
        S = metropolis_step(S,N,T)
        E[i] = energy(S,N)
    return E

```

```python
N_samples = 10000 # no. of samples after thermalization
N_therm = N_samples / 10 # no. of Metropolis algorithm calls for thermalization

Ts = np.arange(0.2, 4.2, 0.2) # temperature range
Us = np.zeros(len(Ts))
Cs = np.zeros(len(Ts))

plt.figure(figsize=(6,6))
observable = 'U' # used for easy plotting
for N_spins in [10,100,1000]:
    for i,T in enumerate(Ts):
        S_init = np.random.choice([-1,1], N_spins) # inital configuration
        E = monte_carlo(S_init,T,N_spins,N_samples,N_therm) # call monte carlo sampler
        Us[i] = np.mean(E) / N_spins # calculate average energy per spin
        Cs[i] = np.var(E) / N_spins / T**2  # calulate specific heat per spin

    # Plotting
    if observable=='U':
        U_th = -(N_spins-1)/N_spins * np.tanh(1/Ts)
        plt.plot(Ts,U_th,'--',label=r'Th., N=%d'%N_spins)
        plt.plot(Ts,Us,label=r"MC, N=%d"%N_spins)
    else:
        C_th = (N_spins-1)/N_spins * (Ts * np.cosh(1/Ts))**(-2)
        plt.plot(Ts,C_th,'--',label=r'Th., N=%d'%N_spins)
        plt.plot(Ts,Cs,label=r"MC, N=%d"%N_spins)

plt.ylim([-0.1,1.0])
plt.legend()
plt.xlabel(r'T')
plt.ylabel(r'%s/N' % observable)
plt.minorticks_on()
plt.grid(which='major', color='#CCCCCC', linestyle='--')
plt.grid(which='minor', color='#CCCCCC', linestyle=':')

fname = '1D_%s_%d' % (observable,N_samples)
plt.savefig('./report/src/%s' % fname, dpi=300)
```

Two-dimensional Ising model

```python
@numba.jit(nopython=True)
def energy_diff(S,N,i,j):
    '''Calculate energy difference to neighbors'''
    l = 0 if i-1 < 0 else S[i-1,j]
    r = 0 if i+1 > N-1 else S[i+1,j]
    u = 0 if j-1 < 0 else S[i,j-1]
    d = 0 if j+1 > N-1 else S[i,j+1]
    return 2*S[i,j]*(r+l+u+d)

@numba.jit(nopython=True)
def energy(S,N):
    '''Calculate energy of configuration S'''
    E = 0
    for i in range(N-1):
        for j in range(N):
            E -= S[i,j]*S[i+1,j]+S[j,i]*S[j,i+1] # see Eq. 1
    return E

@numba.jit(nopython=True)
def magnetization(S):
    '''Calculate magnetization of configuration S'''
    return S.sum()

@numba.jit(nopython=True)
def metropolis_step(S,N,p):
    '''Metropolis algorithm'''

    for _ in range(N*N):
        i = random.randrange(N)
        j = random.randrange(N)
        dE = energy_diff(S,N,i,j)
        if random.random() < np.exp(-dE/T):
            S[i,j] = -S[i,j]
    return S

@numba.jit(nopython=True)
def monte_carlo(S,T,N,N_samples,N_therm):
    '''Repeatedly call Metropolis-Hastings and calculate observables from samples'''

    for _ in range(N_therm): # Thermalize
        S = metropolis_step(S,N,T)

    E = np.zeros(N_samples)
    M = np.zeros(N_samples)
    for i in range(N_samples):
        S = metropolis_step(S,N,T)
        E[i] = energy(S,N)
        M[i] = magnetization(S)
    return E,M,S

Ts = np.arange(0.2,4.2,0.2)
N_samples = 10_000
N_therm = int(N_samples / 10)

Us = np.zeros(len(Ts))
Cs = np.zeros(len(Ts))
Ms = np.zeros(len(Ts))

observable = 'M' # for plotting
plt.figure(figsize=(6,6))
for N in [10,50,100]:
    for i,T in enumerate(Ts):
        S_init = np.random.choice([-1,1], size=(N,N)) # hot start
        E,M,S = monte_carlo(S_init,T,N,N_samples,N_therm)

        # track observables of interest
        Us[i] = np.mean(E) / N**2
        Cs[i] = np.var(E) / N**2 / T**2
        Ms[i] = np.mean(M) / N**2

    if observable == 'U':
        plt.plot(Ts,Us,label=r'MC, N=%d'%N)
    elif observable == 'C':
        plt.plot(Ts,Cs,label=r'MC, N=%d'%N)
    elif observable == 'M':
        plt.plot(Ts,abs(Ms),label=r'MC, N=%d'%N)


if observable == 'M':
    T_c = 2/np.log(1+np.sqrt(2))
    M = lambda T: (1 - np.sinh(2/T)**(-4))**(1/8)
    M_th = np.piecewise(Ts, [Ts < T_c, Ts >= T_c], [M, 0])
    plt.plot(Ts, M_th, '--', c='k', label='Th.')

plt.axvline(T_c, c='red', linestyle='--', label=r'$T_C$')
plt.legend()
plt.xlabel(r'T')
plt.ylabel(r'%s/N$^2$' % observable)
plt.minorticks_on()
plt.grid(which='major', color='#CCCCCC', linestyle='--')
plt.grid(which='minor', color='#CCCCCC', linestyle=':')

fname = '2D_%s_%d' % (observable,N_samples)
plt.savefig('./report/src/%s' % fname, dpi=300)
```