Computational Physics - Exercise 6
1D and 2D Ising model

Christian Gorjaew (354259)
christian.gorjaew@rwth-aachen.de

Julius Meyer-Ohlendorf (355733)
julius.meyer-ohlendorf@rwth-aachen.de

June 15, 2020

1. Introduction

In the following we simulate the behavior of a ferromagnetic system in 1D and 2D at different Temperatures T and extract physical quantities. The system is modeled by the Ising model and the `Metropolis Monte Carlo algorithm` is employed. In the end our results are interpreted in terms of a phase transition.

2. Simulation model and method

The Ising model without an external magnetic field is used. In 1D the Ising model assumes a number N of atomic spins S_i that are arranged on a 1D lattice. Each spin can be in one of two different states, up or down ($S_i = \pm 1$). Therefore, the system can be in 2^N different configurations. For a specific configuration of spins $\{S_1, ..., S_N\}$ the Hamiltonian can be written as:

$$H = -J \sum_{<n,m>} S_n S_m, \tag{1}$$

where the sum runs over all neighboring pairs. As we are modeling a ferromagnetic system the spin-spin interaction parameter J is set to 1. We use free boundary conditions, meaning that lattice points at the end only have one nearest neighbor.
The Ising model is easily extended to the 2D case, where the spins are arranged on a NxN square lattice. Analytic results exist both for the 1D and 2D case. Physical quantities as a function of T can be extracted from the Ising model and can be used to study a phase transition. According to statistical physics the average F of a physical quantity f is extracted as follows:

$$F = \sum_{\{S_1,...,S_N\}} p(S_1, ..., S_N) f(S_1, ..., S_N) = \frac{\sum_{\{S_1,...,S_N\}} f(S_1, ..., S_N) e^{-\beta E(S_1,...,S_N)}}{\sum_{\{S_1,...,S_N\}} e^{-\beta E(S_1,...,S_N)}} \tag{2}$$

One sees, in order to calculate the probability distribution a sum running over all possible configurations needs to be considered, which is not feasible computationally.
To overcome this problem, the `Metropolis Monte Carlo (MMC) algorithm` is employed. For a detailed overview of one MMC step see lecture 13, slide 44. Due to the algorithms design, it only samples a set of configurations Ω, consisting of the most important configurations ("Importance sampling"), which are distributed according to the unknown probability distribution. The average of a physical quantity is then given by:

$$F \approx \frac{1}{\#\Omega} \sum_{\{S_1,...,S_N\} \in \Omega} f(S_1, ..., S_N) \tag{3}$$

In our simulation at first we thermalize the system performing N_{wait} MMC steps at each T (thermalization run). Afterwards we use each thermalized system to perform N_{samples} MMC steps and extract physical properties according to equation (3) (Measurement run).

3. Simulation results

3.1. 1D

The code of this part can be found in the Appendix A.

In the 1D case, we arranged N spins as $N+2$ dimensional arrays with random values ± 1 where the first and last entries were manually set to zero. The latter were introduced for easier handling of the free boundary conditions.

We used a temperature dependent number of thermalization steps N_{wait}. For all evaluated system sizes, $N_{\text{wait}} = 1000$ for $T \geq 1.6$ and $N_{\text{wait}} = 10\,000$ for $T < 1.6$ (note that due to $k_B \equiv 1$, temperatures and energies are dimension-less). The high number of thermalization steps for small T was necessary since the thermalization takes more time due to the larger argument in the Boltzmann factor. For the desired parameters of N_{sample} and N, the average energy and heat capacity were extracted from the measurement run as:

$$\frac{U}{N} = \frac{1}{N}\left(\frac{1}{N_{\text{sample}}}\sum_{i=1}^{N_{\text{sample}}} E_i\right) \qquad \frac{C}{N} = \frac{s_E^2\beta^2}{N} \tag{4}$$

where

$$s_E^2 = \frac{1}{N_{\text{sample}}-1}\sum_{i=1}^{N_{\text{sample}}} (E_i - U)^2 \tag{5}$$

The uncertainties were obtained using[1]

$$\sigma_{U/N} = \frac{1}{N}\frac{s_E}{\sqrt{N_{\text{sample}}}} \tag{6}$$

$$\sigma_{C/N} = \frac{2s_E\beta^2}{N}\sigma_{s_E} = \sqrt{\frac{2}{N_{\text{sample}}}}\frac{s_E^2\beta^2}{N} \tag{7}$$

where the uncertainty on the variance of E, $\sigma_{s_E} = s_E/\sqrt{2N_{\text{sample}}}$ was used and was propagated onto C/N. Here, the obtained quantities were assumed to be uncorrelated since between each measurement a large amount of lattice updates occurred.

The obtained simulation results and theoretical predictions can be seen in Figure 1. In particular, the theory predicts

$$\frac{U_{\text{theo}}}{N} = -\frac{N-1}{N}\tanh(\beta) \qquad \frac{C_{\text{theo}}}{N} = \frac{N-1}{N}\left(\frac{\beta}{\cosh(\beta)}\right)^2 \tag{8}$$

As one can see, the data follows the theory qualitatively. Still, deviations, especially for low temperatures, are visible which appear to be larger for smaller systems size. Also, the data tends to fluctuate stronger for the case of 10 000 samples. Although at lower temperatures the simulated data does not agree with the analytical result within the calculated error, no systematic patterns are visible. This might be an indicator that we underestimated the uncertainties.

[1] c.f., `https://web.eecs.umich.edu/~fessler/papers/files/tr/stderr.pdf`

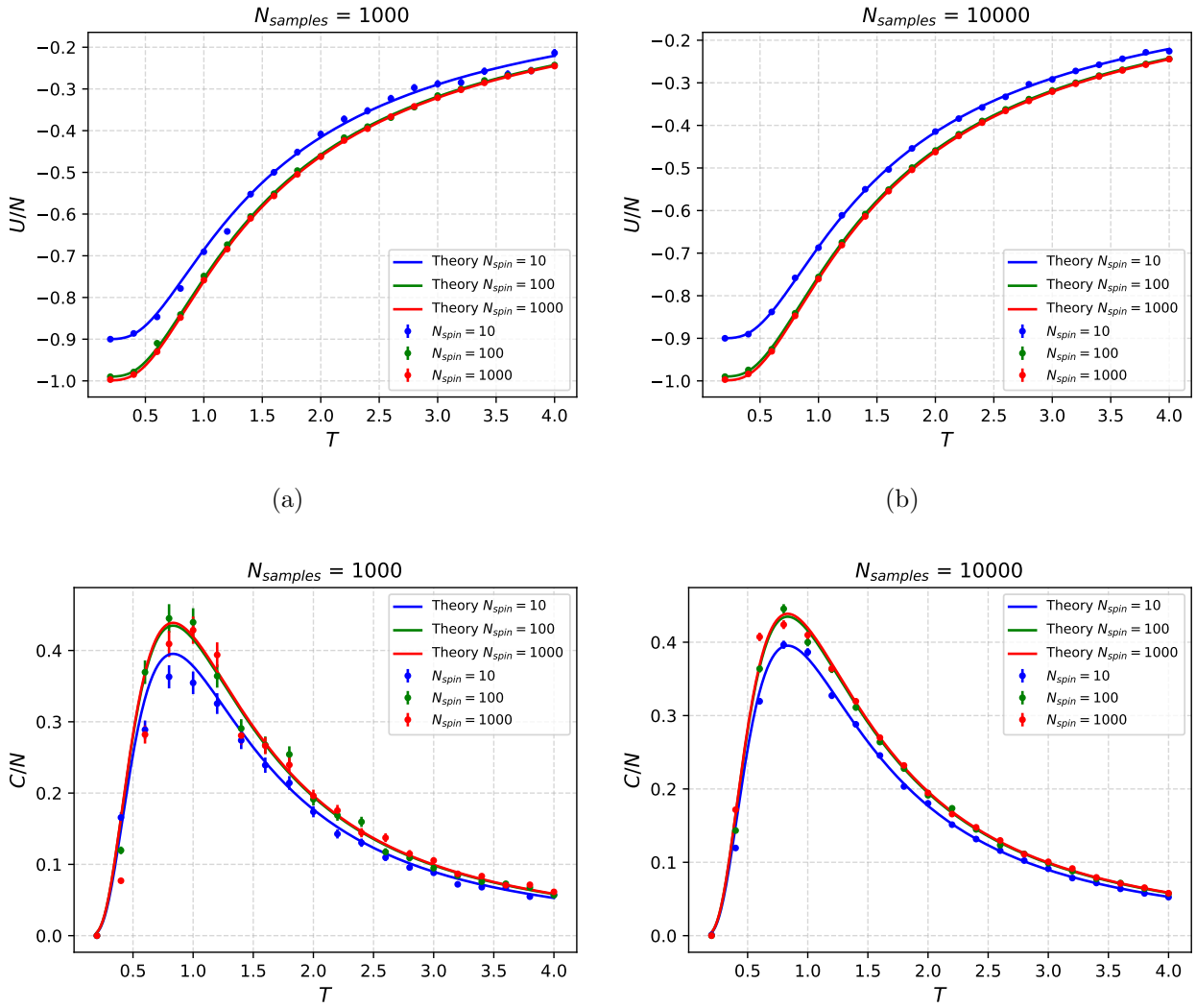


Figure 1: Simulated average energy per spin ((a) and (b)) and heat capacity per spin ((c) and (d)) against temperature T using the MMC method for different numbers of samples and system sizes. Also shown are the theoretical predictions according to equation (8).

3.2. 2D

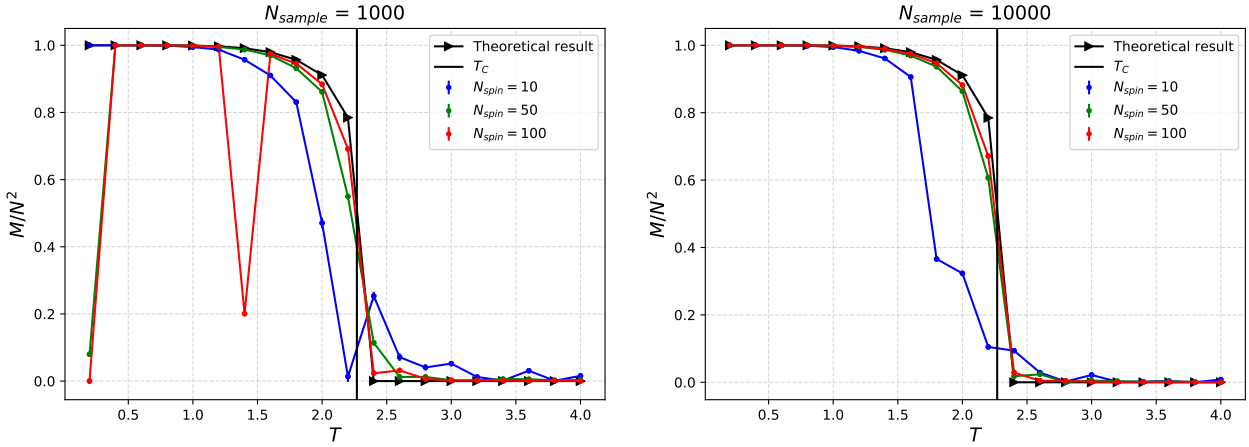
The code of this part can be found in the Appendix B. At first a $(N+2)$ x $(N+2)$ square lattice was randomly initialized with values ± 1. The values of the first and last row and column were set to 0, again for a easier handling of the free boundary conditions.
Like in the 1D case we used a temperature dependent and additionally a system size dependent number of thermalization steps. For systems $N = 10, 50$: $N_{\text{wait}} = 100000$, for $N = 100$: $N_{\text{wait}} = 200000$ for $T \leq 2.6$. And for all systems $N = 10, 50, 100$: $N_{\text{wait}} = 10000$ for $T > 2.6$.
Using the thermalized lattice, we then performed a measurement run performing $N_{\text{sample}} = 1000, 10000$ MMC steps. The energy per spin $\frac{U}{N^2}$ and the average specific heat per spin $\frac{C}{N^2}$ including uncertainties were calculated as in the 1D case (c.f. equations (4) - (7) with $N \to N^2$). The average magnetization per spin $\frac{M}{N^2}$ and its uncertainty were calculated as follows:

$$\frac{M}{N^2} = \frac{1}{N^2}\left(\frac{1}{N_{\text{sample}}} \sum_{i=1}^{N_{\text{sample}}} M_i\right) \qquad \sigma_{M/N^2} = \frac{1}{N^2}\frac{s_M}{\sqrt{N_{\text{sample}}}}, \tag{9}$$

with $M_i = \sum_{n,m=1}^{N} S_{n,m}$ after each N_{sample} step.
Analytic results of the 2D case exist for the average energy U and heat capacity C but are rather complicated. Here, only the theoretical result of the average magnetization per spin $\frac{M}{N^2}$ for an infinite lattice is considered:

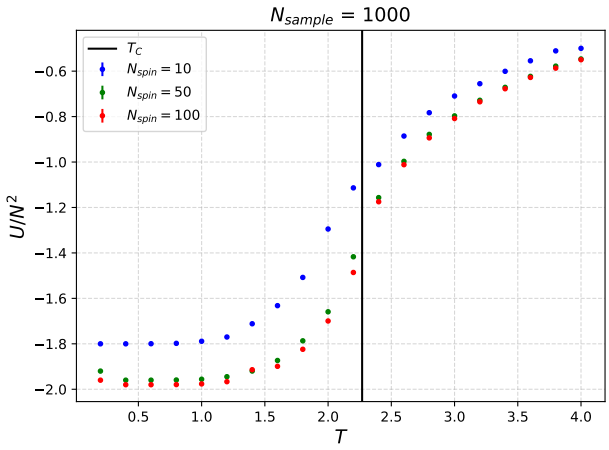
$$\frac{M}{N^2} = \begin{cases} \left(1 - \sinh(2\beta)^{-4}\right)^{\frac{1}{8}} & \text{if } T < T_C \\ 0 & \text{if } T > T_C \end{cases} \tag{10}$$

Our results including the theoretical result of the average magnetization per spin can be seen in Figure 2.

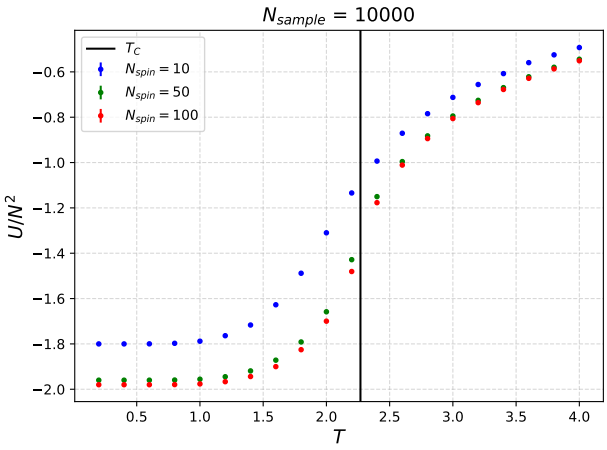


(a)

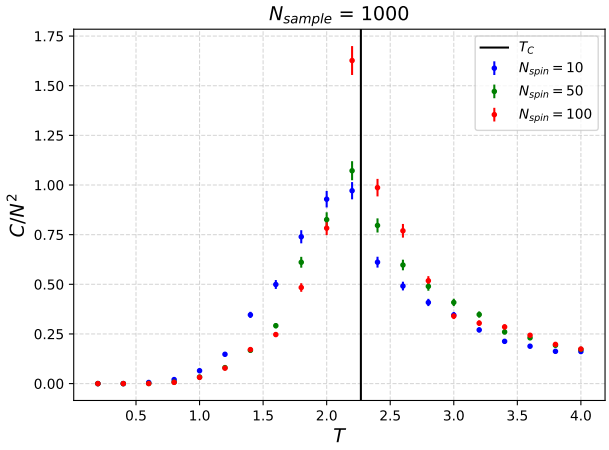
(b)



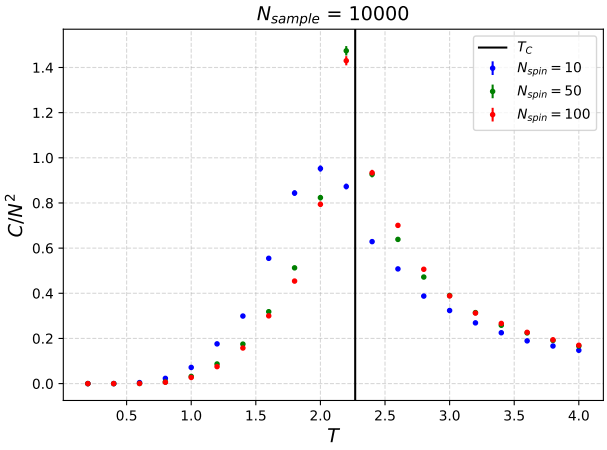
(c)



(d)



(e)



(f)

Figure 2: Extracted quantities from the measurement run for $N_{\text{sample}} = 1000$ (left) and $N_{\text{sample}} = 10000$ (right). The vertical black line marks the critical temperature $T_C = 2.27$.(a) and (b): Average magnetization per spin $\frac{M}{N^2}$ as a function of the temperature including theoretical result. (b) and (c): Average energy per spin $\frac{U}{N^2}$ as a function of the temperature. (e) and (f): Average specific heat per spin $\frac{C}{N^2}$ as a function of the temperature.

In theory, for high temperatures thermal fluctuations dominate, so that the spins are not oriented in one preferred direction. Thus the net magnetization per spin is $\frac{M}{N^2} = 0$. For lower temperatures at around a critical temperature $T_C = \frac{2}{ln(1+\sqrt{2})}$ the system moves to an ordered low temperature phase. The spins are predominately aligned resulting in a net magnetization per spin of $\frac{M}{N^2} = 1$. This phase transition is clearly visible for all our three lattice configurations $N = 10, 50, 100$.
For $N_{\text{sample}} = 1000$ and system sizes $N = 50, 100$ the extracted average magnetization follows the theoretical result rather well. At low temperatures three outliers exist. A possible reason for that could be an unfavorable random initial lattice configuration which made it difficult for the system to reach the thermalized state. Thus, in those cases a larger number of thermalization steps N_{wait} might have been helpful. For system size $N = 10$ our simulated result does follow the the theoretical result very well, especially in the transition region close to T_C. The deviations could be explained by the fact that the theoretical result was obtained for an infinite lattice. Compared to this, a system size of $N = 10$ is rather small and the ratio of lattice points sitting at the edges of the lattice (1 or 2 neighbors) to the points sitting inside of the lattice (4 neighbors) is big. Therefore, edge lattice points have a higher influence on the result compared to the infinite lattice configuration. As one would expect our results follow the theoretical result even better for $N_{\text{sample}} = 10000$. But again larger deviations are observed for the smallest system $N = 10$.
Our results of the average energy per spin also clearly depict the behavior of a phase transition. For $T < T_C$ the systems are in the ground state with the lowest energy. At around T_C the energies reach their inflection point and then start to saturate at temperatures $T > T_C$. The energy curve for the smallest system $N = 10$ is offset in the vertical direction compared to the other systems. This is due to the small system size (high ratio of lattice points on edges to lattice points inside). Qualitatively our results of the specific heat per spin also follows the expected behavior. At around T_C the specific heat diverges ($C = \frac{\partial U}{\partial T}$).

4. Conclusion

Ferromagnetic systems in 1D and 2D were investigated using the Ising model and `Metropolis Monte Carlo algorithm`. In the 1D case our results follow the theory qualitatively. At smaller temperatures deviations are visible, indicating that uncertainties might be underestimated. In 2D the extracted quantities show the characteristics of a phase transition. Especially, for a higher sample number $N_{\text{sample}} = 1000$ our results of the average magnetization per spin follow the theoretical result quite well. Only the smallest system $N = 10$ shows larger deviations in the region of T_C. This is most likely the result of the small system size itself.

A. Code for the 1D Ising model

```
"""
Python version 3.7.7

@author: Chritian Gorjaew, Julius Meyer-Ohlendorf
"""

import numpy as np
import matplotlib.pyplot as plt
from numba import njit
from tqdm import tqdm

@njit  # gives significant speedup by precompiling the following function
def MMC_step(lattice, Energy, beta):
    '''
    Performs one step of the MMC algorithm on the lattice. One MMC step in this
    context are N lattice updates at random positions.

    :param lattice: lattice used to perform one MMC step
    :type lattice: ndarry shape (N+2,), shape due to free BC
    :param Energy: Energy of the system in the given lattice configuration
    :type Energy: float
    :param beta: inverse temperature
    :type beta: : float

    :returns: Energy of the lattice configuration after N updates
    :rtype: ndarray
    '''

    N = lattice.shape[0] - 2

    for k in range(N):

        ridx = np.random.randint(1, N+1)
        idx_l = ridx - 1
        idx_r = ridx + 1

        delta_E = 2 * lattice[ridx] * (lattice[idx_l] + lattice[idx_r])
        q = np.exp(- beta * delta_E)
        r = np.random.uniform(0., 1.)

        if q > r:
            lattice[ridx] *= -1
            Energy += delta_E

    return Energy


'''
############### Performing simuation for desired parameters ###############
'''
simulation = True
evaluation = True
path = "./"

T = np.arange(0.2, 4.2, 0.2)
beta = 1. / T

N_spin = np.array([10, 100, 1000])
N_samples = np.array([1000, 10000])

N_wait_cold = 10000  # temperature dependent number of thermalization steps
```

1

```
N_wait_warm = 1000

if simulation:

    for n in tqdm(N_spin):

        for N in N_samples:

            for i in range(beta.shape[0]):
                b = beta[i]
                lattice = np.random.choice([-1,1], size=(n + 2,))
                lattice[0], lattice[-1] = 0, 0
                E_wait = -np.sum(lattice[1:-2] * lattice[2:-1])

                N_wait = N_wait_warm if T[i] >= 1.6 else N_wait_cold

                # thermalization
                for j in range(N_wait):
                    E_wait = MMC_step(lattice, E_wait, b)

                E = np.zeros(N + 1)  # here the energies of the production run
                                     # are stored
                E[0] = E_wait
                for k in range(N):
                    E[k+1] = MMC_step(lattice, E[k], b)

                # energies are stored and further analyzed below
                save_path = path + "N{0}_Nsample{1}_T{2:.1f}.npz".format(n, N,
                                                                         T[i])
                np.savez(save_path, Energy = E)

'''
############ Evaluation and plotting #############
'''
if evaluation:

    # Arrays storing qunatites for different temperatures, system and samples
    # sizes
    U_all = np.empty((len(N_spin), len(N_samples), len(T)))
    U_all_std = np.empty((len(N_spin), len(N_samples), len(T)))
    C_all = np.empty((len(N_spin), len(N_samples), len(T)))
    C_all_std = np.empty((len(N_spin), len(N_samples), len(T)))

    # Analytical result
    T_linspace = np.linspace(T[0], T[-1], 1000)
    beta_linspace = 1 / T_linspace

    U_theo = - np.tanh(beta_linspace)[:,np.newaxis] * (N_spin - 1) / N_spin
    C_theo = ((beta_linspace / np.cosh(beta_linspace))**2)[:, np.newaxis]
    C_theo *= (N_spin - 1) / N_spin

    colors = ["b", "g", "r"]
    marker = ["."]


    for j, N in enumerate(N_samples):
        fig_u, ax_u = plt.subplots(1,figsize=(5,4))
        fig_c, ax_c = plt.subplots(1,figsize=(5,4))

        ax_u.set_xlabel("$T$", fontsize=12)
        ax_u.set_ylabel("$U / N$", fontsize=12)
```

```
ax_u.set_title("$N_{samples}$ = " + "{}".format(N))

ax_c.set_xlabel("$T$", fontsize=12)
ax_c.set_ylabel("$C / N$", fontsize=12)
ax_c.set_title("$N_{samples}$ = " + "{}".format(N))

for i, n in enumerate(N_spin):

    ax_u.plot(T_linspace, U_theo[:,i], color=colors[i],
              label="Theory $N_{spin} = $" + str(n))

    ax_c.plot(T_linspace, C_theo[:,i], color=colors[i],
              label="Theory $N_{spin} = $" + str(n))


    for k in range(T.shape[0]):

        save_path = path + "N{0}_Nsample{1}_T{2:.1f}.npz".format(n, N,
                                                                 T[k])

        data = np.load(save_path)
        E_tmp = data["Energy"][1:]
        data.close()

        # Here the physical qunatities are calculated as described in
        # in the report
        U_all[i,j,k] = np.mean(E_tmp)
        U_all_std[i,j,k] = np.std(E_tmp, ddof=1) / np.sqrt(N)
        C_all[i,j,k] = np.var(E_tmp, ddof=1) * beta[k]**2 / n

        U_all[i,j,k] /= n
        U_all_std[i,j,k] /= n
        C_all_std[i,j,k] = np.sqrt(2./(N)) * np.var(E_tmp, ddof=1)
        C_all_std[i,j,k] *= beta[k]**2 / n

    label = "$N_{spin} = $" + str(n)
    fmt = colors[i] + marker[0]
    ax_u.errorbar(T, U_all[i,j,:], yerr=U_all_std[i,j,:],
                  fmt=fmt, label=label)
    ax_c.errorbar(T, C_all[i,j,:], yerr=C_all_std[i,j,:],
                  fmt=fmt, label=label)

ax_u.legend(fontsize=8)
ax_u.grid(linestyle="--", alpha=0.5)
ax_c.legend(fontsize=8)
ax_c.grid(linestyle="--", alpha=0.5)
fig_u.tight_layout()
fig_c.tight_layout()
fig_u.savefig(path + "U_N{}_1D.pdf".format(N))
fig_c.savefig(path + "C_N{}_1D.pdf".format(N))
fig_u.show()
fig_c.show()
```

B. Code for the 2D Ising model

```
import numpy as np
from numba import njit
import matplotlib.pyplot as plt

def calc_E(lattice, N):
    '''
    Calculates energy of 2D lattice

    :param lattice: lattice used to calculate energy
    :type lattice: ndarry shape (N+2, N+2), shape due to free BC
    :param N: number of spins
    :type N: : int

    :returns: calculated energy
    :rtype: float
    '''
    E = 0
    # sweeping through rows
    E -= np.sum(np.multiply(lattice[:,1:N], lattice[:,2:N+1]))
    # sweeping through columns
    E -= np.sum(np.multiply(lattice[1:N], lattice[2:N+1]))

    return(E)


@njit
def MMC_step(lattice, beta, N):
    '''
    Performs one step of the MMC algorithm on the lattice, so that resulting
    configurations are distributed according to the unknown distribution

    :param lattice: lattice used to perform one MMC step
    :type lattice: ndarry shape (N+2, N+2), shape due to free BC
    :param beta: inverse temperature
    :type beta: : float
    :param N: number of spins
    :type N: : int

    :returns: lattice
    :rtype: ndarray
    '''

    for k in range(N**2):
        # picking index of spin
        i = np.random.randint(1, N+1)
        j = np.random.randint(1, N+1)

        delta_E =  2 * lattice[i][j] * (lattice[i-1][j] + lattice[i+1][j] +
                                        lattice[i][j-1] + lattice[i][j+1])

        # deciding if flipped or not
        if delta_E < 0:
            lattice[i][j] *= -1

        else:
            q = np.exp(-beta * delta_E)
            r = np.random.uniform(0., 1.)
            if q > r:
                lattice[i][j] *= -1

    return(lattice)
```

1

```
# Paramters
##########################################################
##########################################################

# Parameters of the script
simulation = False
saving = False
evaluation = True
errorbar = True

# Parameters of the simulation
N_arr = np.array([10, 50, 100])
N_sample_arr = np.array([1000, 10000])
T_arr = np.arange(0.2, 4.2, 0.2)
beta_arr = 1 / T_arr

N_wait1 = 100000
N_wait2 = 200000

# Relaxation and measurement run
##########################################################
##########################################################
if simulation:

    for N in (N_arr):
        print('N:', str(N))

        for N_sample in (N_sample_arr):
            print('N_sample:', str(N_sample))

            for T in (T_arr):
                print('T:', str(T))
                beta = 1 / T

                # Initialization lattice
                # Due to free BC it is easier to add a ring of zeros around the
                # considered lattice for later calculations

                lattice = np.random.choice([1,-1], size=(N+2, N+2))
                lattice[:, 0] = 0
                lattice[:, N+1] = 0
                lattice[0, :] = 0
                lattice[N+1, :] = 0

                # Thermalization run:
                # Choosing N_wait depending on N and T
                if (N == 10) or (N == 50):
                    N_wait = N_wait1 if T < 2.6 else 10000
                else:
                    N_wait = N_wait2 if T < 2.6 else 10000

                for i in range(N_wait):
                    lattice = MMC_step(lattice, beta, N)

                # Measurement run:
                M_arr = np.zeros(N_sample)
                E_arr = np.zeros(N_sample)

                for i in range(N_sample):
                    lattice = MMC_step(lattice, beta, N)
```

```
                    M_arr[i] = np.sum(lattice[1:N+1, 1:N+1])
                    E_arr[i] = calc_E(lattice, N)

                if saving:
                    np.savez('./simulation/2D/N{0}_Nsample{1}_T{2:.1f}_{3}_{4}.npz'
                             .format(N, N_sample, T, N_wait1, N_wait2), E=E_arr, M=M_ar


# Loading and plotting data from measurement
##########################################################
##########################################################
if evaluation:

    # three arrays holding mean and std of  U/N**2, C/N**2 and M/**2
    # for all parameters

    U_arr = np.zeros((len(N_arr), len(N_sample_arr), len(T_arr)))
    U_std_arr = np.zeros((len(N_arr), len(N_sample_arr), len(T_arr)))

    C_arr = np.zeros((len(N_arr), len(N_sample_arr), len(T_arr)))
    C_std_arr = np.zeros((len(N_arr), len(N_sample_arr), len(T_arr)))

    M_arr = np.zeros((len(N_arr), len(N_sample_arr), len(T_arr)))
    M_std_arr = np.zeros((len(N_arr), len(N_sample_arr), len(T_arr)))

    # theoretical values for M
    M_theo = np.zeros(len(T_arr))
    Tc = 2 / (np.log(1 + np.sqrt(2)))
    for i,T in enumerate(T_arr):
        if T < Tc:
            M_theo[i] = (1 - np.sinh(2 * beta_arr[i])**(-4))**(1/8)

    for j, N_sample in enumerate(N_sample_arr):

        fig_u, ax_u = plt.subplots(1)
        fig_c, ax_c = plt.subplots(1)
        fig_m, ax_m = plt.subplots(1)

        ax_m.plot(T_arr, M_theo, color='k', marker='>', label='Theoretical result')

        ax_u.set_xlabel("$T$ [au]", fontsize=14)
        ax_u.set_ylabel("$U / N^{2}$", fontsize=14)
        ax_u.set_title("$N_{sample}$ = " + "{}".format(N_sample), fontsize=14)
        ax_c.set_xlabel("$T$", fontsize=14)
        ax_c.set_ylabel("$C / N^{2}$", fontsize=14)
        ax_c.set_title("$N_{sample}$ = " + "{}".format(N_sample), fontsize=14)
        ax_m.set_xlabel("$T$", fontsize=14)
        ax_m.set_ylabel("$M / N^{2}$", fontsize=14)
        ax_m.set_title("$N_{sample}$ = " + "{}".format(N_sample), fontsize=14)

        colors = ["b", "g", "r"]
        marker = "."

        for i, N in enumerate(N_arr):

            for k, T in enumerate(T_arr):
                beta = 1 / T
                data = np.load('./simulation/2D/N{0}_Nsample{1}_T{2:.1f}_{3}_{4}.npz'
                               .format(N, N_sample, T, N_wait1, N_wait2))
                E = data['E'][:]
                M = data['M'][:]
```

```
            data.close()

            U = np.mean(E)
            U2 = U**2
            E2 = E**2
            E_std = np.std(E, ddof=1)

            # calculating mean and std
            U_arr[i,j,k] = U / N**2
            U_std_arr[i,j,k] = E_std / np.sqrt(N_sample) / N**2

            C_arr[i,j,k] = np.var(E, ddof=1) * beta**2 / N**2
            C_std_arr[i,j,k] = np.sqrt(2./(N_sample)) * np.var(E, ddof=1)
            C_std_arr[i,j,k] *= beta**2 / N**2

            M_arr[i,j,k] = np.abs(np.mean(M)) / N**2
            M_std_arr[i,j,k]  = np.std(M, ddof=1) / np.sqrt(N_sample) /N**2

        label = '$N_{spin} = $' + str(N)
        fmt = colors[i] + marker
        color = colors[i]

        if errorbar:
            ax_u.errorbar(T_arr, U_arr[i,j,:], yerr=U_std_arr[i,j,:],
                          fmt=fmt, label=label)
            ax_c.errorbar(T_arr, C_arr[i,j,:], yerr=C_std_arr[i,j,:],
                          fmt=fmt, label=label)
            ax_m.errorbar(T_arr, M_arr[i,j,:], yerr=M_std_arr[i,j,:],
                          fmt=fmt, label=label)
            ax_m.plot(T_arr, M_arr[i,j,:], color=color, marker=marker)
        else:
            ax_u.plot(T_arr, U_arr[i,j,:], color=color,
                      marker=marker, label=label)
            ax_c.plot(T_arr, C_arr[i,j,:], color=color,
                      marker=marker, label=label)
            ax_m.plot(T_arr, M_arr[i,j,:], color=color,
                      marker=marker, label=label)

ax_u.axvline(x=Tc, color='k', label='$T_{C}$')
ax_c.axvline(x=Tc, color='k', label='$T_{C}$')
ax_m.axvline(x=Tc, color='k', label='$T_{C}$')
ax_u.legend(fontsize=10)
ax_u.grid(linestyle="--", alpha=0.5)
ax_c.legend(fontsize=10)
ax_c.grid(linestyle="--", alpha=0.5)
ax_m.legend(fontsize=10)
ax_m.grid(linestyle="--", alpha=0.5)
fig_u.tight_layout()
fig_c.tight_layout()
fig_m.tight_layout()
fig_u.savefig('./simulation/2D/U_errorbar_{0}_{1}_{2}_Nsample{3}.pdf'.format(str
fig_c.savefig('./simulation/2D/C_errorbar_{0}_{1}_{2}_Nsample{3}.pdf'.format(str
fig_m.savefig('./simulation/2D/M_errorbar_{0}_{1}_{2}_Nsample{3}.pdf'.format(str
```