Monte Carlo Simulations for Two-dimensional Ising System

Qing Zhou, s2501597

April, 2020

Abstract

In this report, a two dimensional Ising model was simulated, using the Metropolis algorithm. Its thermal properties were investigated after the system reached equilibrium. To estimate the errors we implemented the blocking method. It was found that the system underwent a "phase transition" near the critical temperature, yielding a sudden cutoff for the average magnetisation, and the peaks for autocorrelation time, magnetic susceptibility and heat capacity. The behaviours of the thermal quantities are expected and in agreement with those found in other literatures[1][2][3].

1 Introduction

Monte Carlo methods are a broad class of computational algorithms which rely on repeatedly sampling to estimate different properties. Due to its innate stochastic property it is most useful in observing thermodynamic properties of systems which are impossible to analyse using other approaches. The two-dimensional Ising model consists of a systems of spins on a lattice. By implementing the Monte Carlo method, we can approximate the interaction of the spins, by a random process with some deterministic probability, and gauge the evolution of the system.

This report is laid out as follows: in section 2 we introduce the basics of Monte Carlo simulations and its applications; in section 3 the subject of this project – two dimensional Ising model is described; section 4 includes all the properties we are to analyse, and associated properties; in section 5 we demonstrate our observational results and discuss their implications; lastly in section 6, we present our findings in this project and propose possibilities for further investigations.

2 Monte Carlo Simulation

The general idea of Monte Carlo simulation is to use the randomness of the process to interpret the deterministic principle of the system to be observed. the main idea of this method is based on vastly random sampling and computational analysis. The evolution of the system can be obtained by implementing the so-called Metropolis algorithm.

2.1 Metropolis Algorithm

Metropolis algorithm is an MCMC method widely applied in statistics and statistical physics, for obtaining a sequence of random deviates, where using direct sampling from the probability distribution is difficult. In this project, we assume that the time evolution of the Ising model satisfies the following probability distribution

$$p(x) \propto e^{-\beta H(x)} \tag{1}$$

where x is spin and H is the corresponding Hamiltonian. The generation routine of this algorithm can be understood as follows

1. Start with an initial state of the system of size $N \times N$ at $\boldsymbol{x}_0$, where $x_i \in \boldsymbol{x}_0$ represents a individual spin;
2. randomly select a spin state $x_i \in \boldsymbol{x}_0$, and compute its energy H_i given by (8);
3. Now also consider its flipped state x'_i, where $x'_i = -x_i$. Compute the corresponding Hamiltonian H'_i;
4. Compute the difference in Hamiltonian $\Delta H_i = H'_i - H_i$, if $\Delta H < 0$, we accept this move $x_i \to x'_i$; otherwise we accept it with a probability given by (1).

The Monte Carlo time step here is defined as follows: for each $x_i \in \boldsymbol{x}$, it has $1/N^2$ probability to be selected for a flip attempt. And naturally we have 1 time step = $N \times N$ flip attempts.

2.2 Accuracies and Statistics

Why do we choose Monte Carlo method for our project? Because it outperforms any other methods. Consider the following case where we have a one-dimensional integral, which is similar to some expectation value we are to observe in our system

$$\langle A \rangle = \int_{-L}^{L} P(x)A(x)dx, \qquad \int_{-L}^{L} P(x)dx = 1 \tag{2}$$

where $P(x)$ is an arbitrary probability distribution. The expectation value of A can be obtained by random sampling M deviates

$$\langle A \rangle \approx \frac{2L}{M} \sum_{i=1}^{M} P(x_i)A(x_i) \tag{3}$$

However, if our $P(x)$ is sharply peaked in a region but flat elsewhere, like a delta function, then the fluctuations of this expectation value will be huge. To circumvent that we can instead do the following: random sampling according to some other probability distribution $W(x)$. Now the sampling is done in the infinitesimal range $[x, x + dx]$, instead of $[-L, L]$. The expectation value can then be recalculated as

$$\langle A \rangle \approx \frac{1}{M} \sum_{i=1}^{M} \frac{P(x_i)}{W(x_i)} A(x_i) \tag{4}$$

resulting in a fluctuation of

$$\sigma_A^2 = \int_{-L}^{L} \left(\frac{P(x)}{W(x)} A(x) - \langle A \rangle \right)^2 W(x)dx \tag{5}$$

which can in principle be reduced by carefully choosing $W(x)$. In practice it is not possible to reduce the fluctuations by choosing the optimal $W(x)$, except for cases where $P(x)$ has much larger fluctuations than $A(x)$, then we can simply set $W(x) = P(x)$, yielding an expectation value of

$$\langle A \rangle = \frac{1}{M} \sum_{i=1}^{M} A(x_i) \tag{6}$$

with fluctuations

$$\sigma_A^2 = \int_{-L}^{L} [A(x) - \langle A \rangle]^2 P(x)dx \tag{7}$$

3 Two-dimensional Ising Model

Ising model is one of the simplest and most famous models for an interacting system. It was originally proposed by Ising in his Ph.D dissertation, as a model for ferromagnetism. When a collection of spins are in alignment, so that their associated magnetic moments all point in the same direction, it is expected that a net magnetic moment is macroscopic in size. Consider a lattice of size $N \times N$, with a spin of either 1 or -1 on each site. Then the interacting Hamiltonian is

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} s_i s_j - H \sum_i s_i \tag{8}$$

The first term represents the interaction of a spin with its four neighbouring spins, and the second term accounts for an external magnetic field, J is a coupling constant. For simplicity we set $J = 1$, $H = 0$, so we have a simple system with internal interactions only. This formula has the following implications: the first term shows that the energy is lowered when neighbouring spins are in alignment due to the *Pauli exclusion principle*. Since electrons cannot occupy the same quantum state, two electrons with parallel spins cannot come close to each other in space. In this project, a two-dimensional Ising model on a square lattice of size $N \times N$ is used, with periodic boundary conditions applied. The system consists of spins of value 1 or -1, representing spin up or spin down. Three ways of initialisation are implemented: i. random initialisation (corresponds to infinite temperature); ii. with spins all pointing up (corresponds to zero temperature); iii. and with spins all pointing down (corresponds to zero temperature). For the system a critical temperature exists, below which spontaneous magnetization happens. This critical temperature can be determined analytically from

$$\frac{k_B T_c}{J} = \frac{2}{\ln\left(1+\sqrt{2}\right)} \approx 2.269 \tag{9}$$

For temperatures greater than the critical temperature T_c, the system is paramagnetic; for $T < T_c$, the system is ferromagnetic and the spin has two solutions ± 1. It is thus expected to see a peak in the equilibration time around the critical temperature, for different setups of temperatures. One caveat is the behaviour of the system at low temperatures ($T < T_c$). The Metropolis algorithm calculates the energy gain for a given flip attempt, by comparing the energy of the spin itself with its neighbours. A flip is granted when the thermal energy computed from the Boltzmann distribution is cool enough for the spin to join its neighbouring alignment ($\Delta H < 0$). The issue here is not really the orientation of each individual spin, yet the sum of each magnetic moment is quantised. At very low temperatures ($T \ll T_c$), the relevant states of a true ferromagnet are "magnons" in which all dipoles are nearly parallel and a single unit of opposite alignment is spread across many dipoles.

4 Thermodynamic Properties

There are a few thermodynamic quantities we are interested in and thus should be kept track of throughout the whole simulation. Perhaps one of the most important question to ask, is how should we define the Markov Chain time step? To answer this question we must go back to the definition of the Markov Chain Monte Carlo (MCMC) method and the main problems it aims to address. MCMC are created to address multi-dimensional problems, by creating samples from a continuous random variable. In general, an ensemble of chains is developed, starting from a set of points arbitrarily chosen and sufficiently distant from each other. After some "random walks" the Markov Chain should by construction has some desired distribution in its equilibrium state. For this we usually require the step to have a detailed balance, so that our Markov Chain can have desired properties. Regarding the Metropolis algorithm to update the system, by construction we have naturally two ways of choosing the Markov Chain step:

1. One Markov Chain step is simply one flip attempt;
2. One Markov Chain step is one sweep across the lattice.

In the second case, we simply consider a step to be a sequence of single updates, with each verifying a "detailed balance". The motivation we that we have chosen the second scheme is that, the state generated by one flip is not sufficient for the system to be independent from its past state, i.e., a degree of correlation exists which relate its future state to past state. To circumvent that we introduce the *autocorrelation time*, so that the states are "disentangled" are measurements can then be carried out.

Once equipped with the above information we can journey on to the properties we want to measure. The first is the total energy of the system, which is simply the sum of the Hamiltonians over all spins

$$E_{tot} = \frac{1}{2}\sum_i \mathcal{H}_i \tag{10}$$

where $\mathcal{H}$ is given by (8). The factor $\frac{1}{2}$ is there so that the interaction between pairs $s_i s_j$ and $s_i s_i$ is not considered twice. Meanwhile it is also useful to keep track of the total magnetisation

$$M = \sum_i s_i \tag{11}$$

of the system. As is addressed above, we should really wait for the Markov Chain to be equilibrated to measure the thermal properties of the system. Hence we first analyse the average magnetisation (the magnetisation per spin), and see how long it takes for its variance to stabilise. Once we have stabilised average magnetisation we can compute its autocorrelation function (in discrete form)

$$\chi(t) = \frac{1}{t_{\max}-t}\sum_{t'=0}^{t_{\max}-t} m(t')m(t'+t) - \frac{1}{t_{\max}-t}\sum_{t'=0}^{t_{\max}-t} m(t') \times \frac{1}{t_{\max}-t}\sum_{t'=0}^{t_{\max}-t} m(t'+t) \tag{12}$$

The autocorrelation function gives us a measure of m at different times: if $m(t')$ and $m(t'+t)$ are indeed fluctuating in the same direction, then the correlation gives a value of 1; if we sum over all possible t' and average the sum out, χ will be positive if on average the fluctuations are correlated. For the Ising model simulated with Metropolis algorithm, if we measure the average magnetisation in two successive MC steps, then obviously the time is not enough for the two states to be completely "disentangled", thus yielding a positive value for χ. On the other hand if we wait long enough for the system to evolve and completely forget its previous state, the a correlation for this time period would be practically zero. The autocorrelation falls off quite rapidly, obeying approximately by a power law as

$$\chi(t) \approx \chi(0)e^{-t/\tau} \tag{13}$$

where τ here is the correlation time. So for a system evolved for $t = \tau$, the similarity of its current state is only $1/e$ to that of the previous one at $t = 0$. If we want truly independent samples to be drawn, we can have samples drawn from a time interval of some τ. A natural choice for this time interval to achieve statistical independence can be $\delta t = 2\tau$, from the Nyquist-Shannon sampling theorem. And consequently the variance from the mean can be computed from

$$\sigma = \sqrt{\frac{2\tau}{t_{\max}}(\langle m^2\rangle - \langle m\rangle^2)} \tag{14}$$

Next, before we carry on with other properties to observe we must first fix the low temperature issue of the Ising model. For $T < T_c$ the average magnetisation shows a net result of either 1 or -1,depending on which state it chooses to settle first. One way to avoid this is to take the absolute value of the average magnetisation, when the system is at $T < T_c$

$$\langle |m| \rangle = \frac{1}{N}\left\langle \left|\sum_i s_i\right| \right\rangle \tag{15}$$

4.1 The Blocking Method

To determine other quantities with low systematics we introduce a simple method: the blocking method, whose main idea is to take measurements of E and M we made during the simulation, and divide them into groups. We then calculate the susceptibility χ_M and specific heat C from the blocks, instead of from the averages of the individual measurements. Then from the blocks we can calculate the two properties from

$$\chi_M = \frac{\beta}{N}\frac{\partial\langle M\rangle}{\partial H} = \frac{\beta}{N}(\langle M^2\rangle - \langle M\rangle^2) \tag{16}$$

for magnetic susceptibility per spin and

$$C = \frac{\partial E}{\partial T} = \frac{1}{Nk_BT^2}\frac{\partial^2 \ln Z}{\partial \beta^2} = \frac{1}{Nk_BT^2}(\langle E^2\rangle - \langle E\rangle^2) \tag{17}$$

5 Measurements and Observations

The setup of the simulation is as follows: the temperature range was set to be in the range $[1, 4]$, with a step of 0.2. For each temperature, we ran 8 simulations, for temperatures $T > 2$, we set $t_{\max}$ to be 10000; otherwise we have $t_{\max} = 1000$. We take 8 simulations at $T = 1$ for example. At this temperature the time needed for the system to settle down is relatively shorter, so for our lattice of size 50×50, we take $t_{\max} = 1000$. Figure 1 shows in detail how we do data analysis and error estimation.

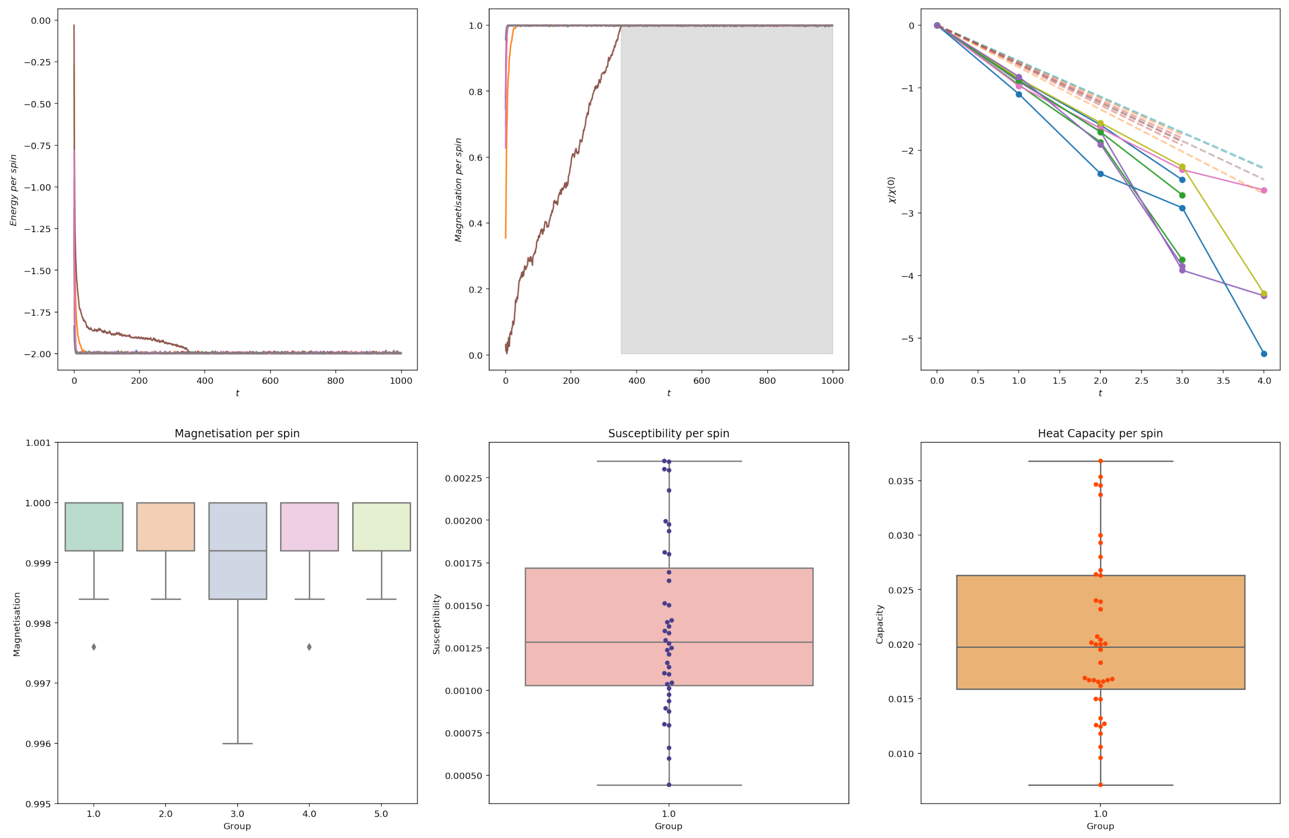


Figure 1: 8 simulations initialised randomly for $T = 1$, with $t_{\max} = 1000$. Top left: the average energy (energy per spin) from $t = 0$ to $t = t_{\max}$, different colours represent different simulation with different initial spin states. Top middle: the corresponding average magnetisation (magnetisation per spin) of the 8 simulations, with the grey region showing where the standard deviation is within 1σ. Top right: the correlation function of the average magnetisation, computed using the values inside the grey region.Here the y-axis is $\chi/chi(0)$, plotted on a logarithmic scale, the x-axis is time. The dashed lines represent the exponential functions $\chi/\chi(0) = -t\tau$, with τ calculated from the discrete summation of $\chi/\chi(0)$. Bottom left: boxplot of the average magnetisation for the last simulation. For each simulation, the measurements of the average magnetisation (inside the grey region) are divided into blocks (groups), each of size 16τ. For this project we take 5 blocks for each simulation, so with 8 simulations per temperature, we have in total 40 statistically independent blocks (the other 35 blocks are not shown on the graph). From the 40 blocks we can have 40 measurements of magnetic susceptibility, and 40 measurements of heat capacity as well (from 40 blocks of energy E, which is not shown). Bottom middle: boxplot of the susceptibility, the dots outside the box are discarded. Bottom right: same but now for the heat capacity per spin.

It can be seen, for $T = 1$ the average magnetisation for the 8 simulation reaches equilibrium after about 400 steps, where the standard deviation is within 1σ. Using the measurements of magnetisation inside the grey region we can obtain the autocorrelation function given by (12). Then to obtain the values of τ we can simply perform a summation on the discrete sequence of $\chi(t)/\chi(0)$. It can be estimated, after time τ, the correlation at $t = \tau$ is roughly a factor of $1/e$ as compared to that at $t = 0$. Since the states within τ are correlated, error inferred straight from the top middle plot will be largely underestimated. We would thus like our measurements to be as independent as possible. Hence we used the blocking method described above. The block size is taken to be 16τ, and for each simulation we take 5 blocks, so for each temperature we have altogether 40 independent blocks, corresponding to 40 (independent) measurements of magnetic susceptibility and heat capacity (from blocks of energy). Measurements too far away from the mean are regarded as ”bad” measurements (denoted by the dots outside the boxes in the bottom middle and bottom right plots), and are thus not included in the calculation.

To demonstrate why we need a longer time epoch to run the simulations for larger values of T, we take the case where $T = 2$ for illustration.

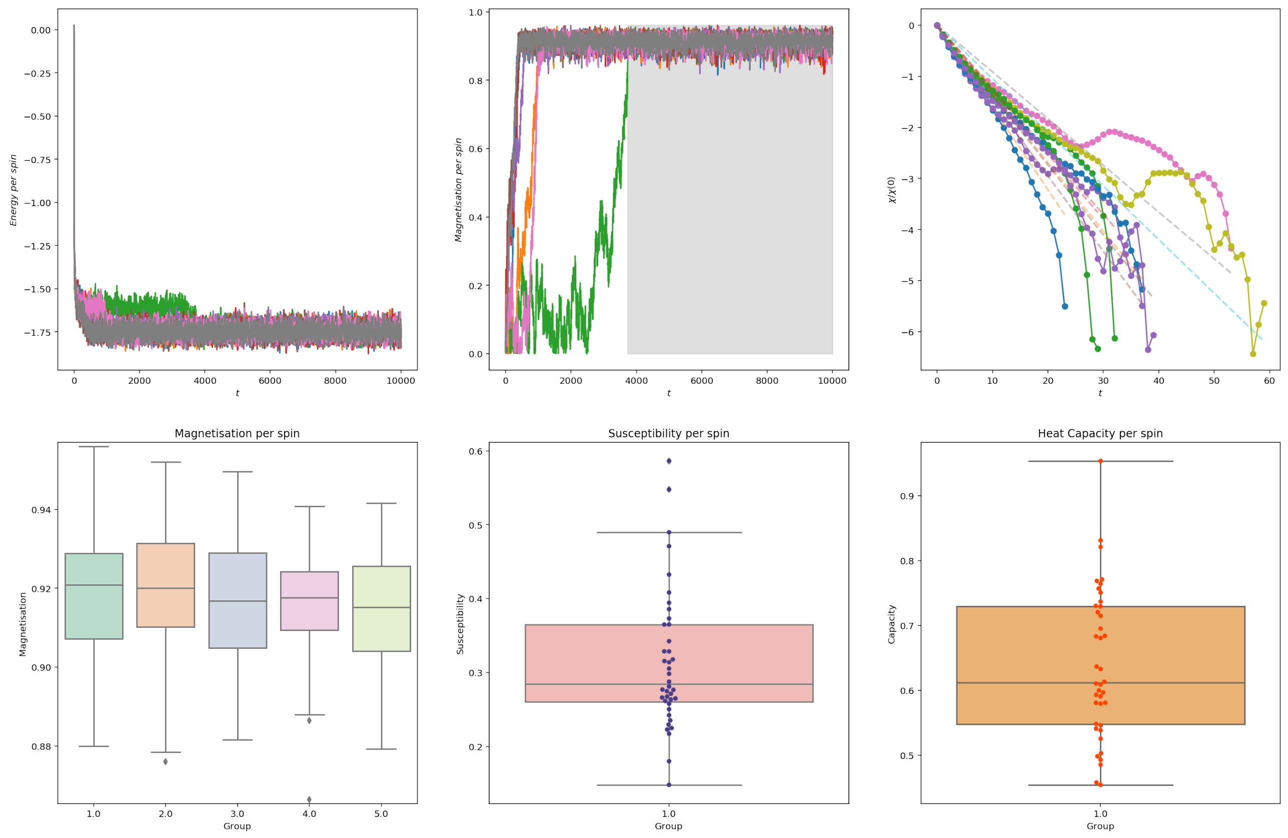


Figure 2: Same as 1, but at $T = 2$ with $t_{\max} = 10000$.

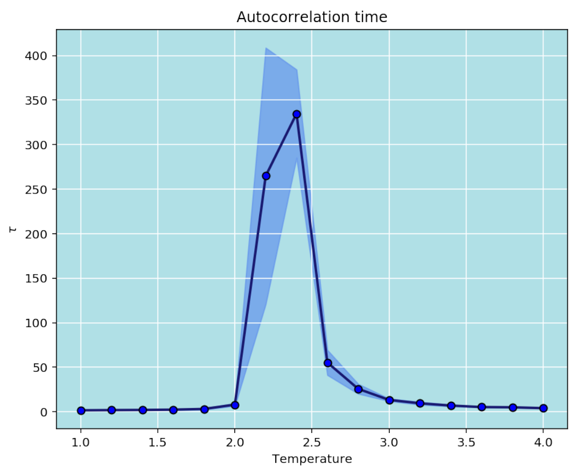
It can be seen, for $T = 2$ which is closer to the critical temperature, it requires more time (~ 4000 steps) for the system to reach equilibrium. The value of τ are roughly 10 times larger than those of $T = 1$.

In the end, the properties as a function of temperature are shown in Figure 3.

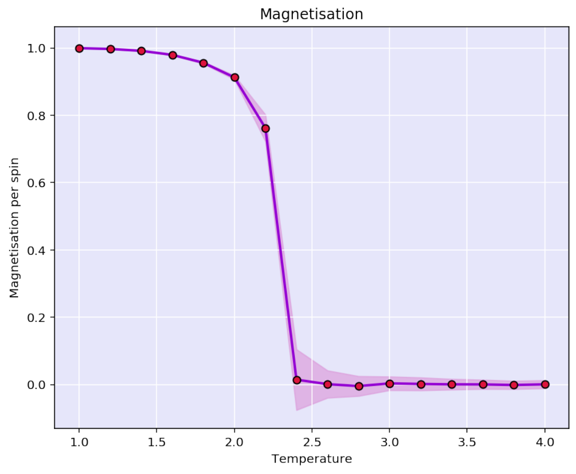
6 Conclusion and Results

It can be seen, with the blocking method the errors (all but the region near the critical temperature) are greatly reduced, even for 8 simulations initialised randomly. How do we expect the magnetisation to behave? For a system of ferromagnetics, we should expect the system undergoing a phase transition near the critical temperature, where the spontaneous alignment is broken. Hence for a system of ferromagnetics, in the absence of an external magnetic field, the average magnetisation should slowly decrease from 1, and the speed at which it decays should increase with larger temperature, and a cutoff is expected to be $\sim T_c$. This is what we see in the top right plot of Figure 3, where the cutoff is around ~ 2.4. Moreover, at low temperatures the system is most likely to quickly converges to a homogeneous magnetisation (seen as in Figure 1 for all the 8 simulations after 400 steps), yet near the critical temperature, the magnetisation is in an "in-between" state, where spins tend to cluster. Thus it takes longer for a system at $T \sim T_c$ to reach equilibrium. On the other hand, at large temperatures, when equilibrium is reached it can be expected the system is in a "well-mixed" state, hence compared to the previous case the time for the system to reach equilibrium is shorter. The behaviour of magnetic susceptibility and heat capacity can be inferred similarly. They are respectively a measure of fluctuations in magnetisation and fluctuations in energy. And for the same reason, when $T \sim T_c$ the system undergoes a phase transition, where the errors are the largest, susceptibility and heat capacity, if well-behaved, should indicate the level of fluctuations correctly. Hence they are both expected to peak near the critical temperature.

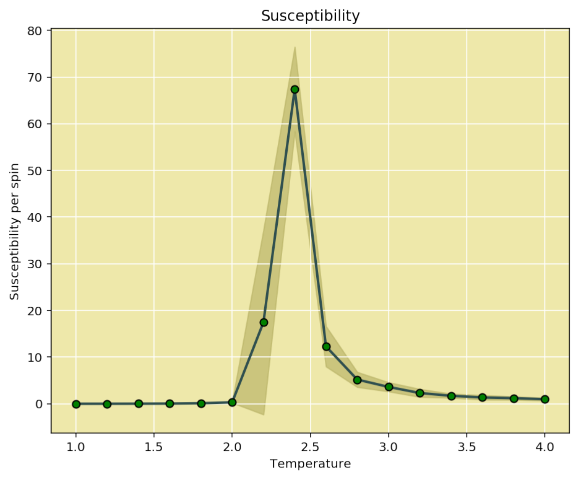
To further investigate the thermodynamic properties of the two dimensional Ising model, it is useful to run the simulation with larger size N. Alternatively it might be useful to implement the Woff algorithm, which is very powerful handling for handling clusters near critical temperature.



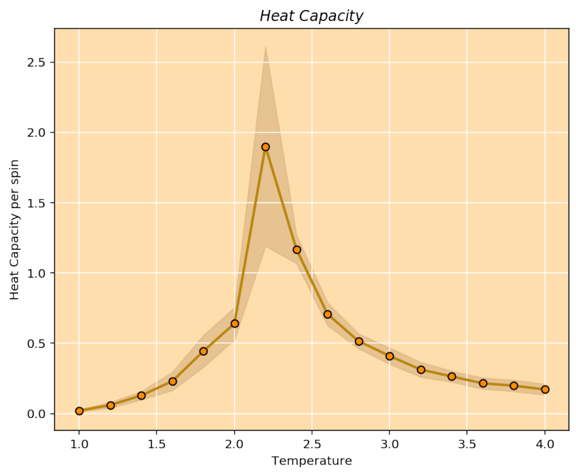
(a) Autocorrelation time of the two dimensional Ising model. The data points are determined by taking the average of the values of τ.



(b) Average magnetisation of the two dimensional Ising model. The data points are determined by taking the mean of the mean of each block.



(c) Susceptibility of the two dimensional Ising model.



(d) Heat capacity of the two dimensional Ising model.

Figure 3: Thermodynamic properties determined for the two dimensional Ising model, simulated using the Metropolis algorithm, details see Figure 1.

References

[1] Newman, M. E. J., and T. Barkema, 1999, Monte Carlo Methods in Statistical Physics (Oxford University Press, Oxford, UK)

[2] Maria I. Dias A., CERN Summer Student Report: Testing Lorentz Invariance Emergence in the Ising Model using Monte Carlo simulations

[3] Richard Fitzpatrick, 2006,The Ising model, retrieved from `http://farside.ph.utexas.edu/teaching/329/lectures/node110.html`

7 Appendix

```python
#!/usr/bin/env python
# coding: utf-8

import numpy as np
from mpi4py import MPI

```

```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

"""Function that computes the Hamiltonian applying the periodic boundary condition.
Argument: spin -- 2d array that stores the spins of a timestep.
"""
def spin_sum(spin):
    spin_pad = np.pad(spin,((1,1),(1,1)),mode='wrap')
    n1 = spin_pad[1:-1,:-2]
    n2 = spin_pad[2:,1:-1]
    n3 = spin_pad[1:-1,2:]
    n4 = spin_pad[:-2,1:-1]
    return -(n1+n2+n3+n4)*spin

"""
Function that implements the Metropolis algorithm to update the spin states of the lattice.
Argument: N -- if N = 3, a lattice of 3 x 3 sites is created.
          T -- temperature.

"""
def H(N,step,T=1,ds=1):
    #critical temperature
    T_crit = 2.269
    #total magnetisation
    tot_mag = np.zeros(int(step/ds))
    #(twice of) total energy
    totE = np.zeros(int(step/ds))
    #initialising the spin states
    spin = np.random.randint(0,2,N*N)
    mask = spin == 0
    spin[mask] = -1
    spin = spin.reshape(N,N)
    x = spin_sum(spin)
    tot_mag[0] = np.sum(spin)
    totE[0] = np.sum(x)
    count = 0

    while count < step*N*N-1:

        #attempt to flip a spin
        ind = np.random.randint(0,N*N)
        i = (ind+1)//N - 1
        j = ind - (i+1)*N -1
        spin1 = spin.copy()
        spin1[i][j] = -1*spin[i][j]
        x1 = spin_sum(spin1)
        diff = x1[i][j]-x[i][j]
        count += 1

        #accept with probability 1 if delta E < 0
        if diff <= 0:
            spin = spin1
            x = x1

            if count%(ds*N*N) == 0:
                tot_mag[int(count/(ds*N*N))] = np.sum(spin)
                totE[int(count/(ds*N*N))] = np.sum(x)


        #if delta E > 0
        else:
            kB = 1
            beta = 1./(kB*T)
            p = np.exp(-beta*diff)
            n = np.random.uniform(0,1)
            #updating spin with probability p
            if n<=p:
                spin = spin1
                x = x1
                #write data per sweep
                if count%(ds*N*N) == 0:
                    tot_mag[int(count/(ds*N*N))] = np.sum(spin)
                    totE[int(count/(ds*N*N))] = np.sum(x)


            else:
                #write data per sweep
                if count%(ds*N*N) == 0:
                    tot_mag[int(count/(ds*N*N))] = np.sum(spin)
                    totE[int(count/(ds*N*N))] = np.sum(x)

    if T > T_crit:
        return totE,tot_mag
    else:
        return totE,np.abs(tot_mag)


#storing the simulations to files
ind = 0
for T in np.arange(1,4.2,0.2):
    for sim in range(40):
        if rank == ind%size:
            print('rank {} run T={}, sim={}'.format(rank, T, sim))
            E, M = H(50,10000,T)

    np.c_[E,M].tofile('/data/dell5/userdir/ucas_students/ZQ/MC_sim1/T=%.2f'%T+'_sim%d'%(sim+1)+'.bin')
        ind += 1


# mpirun -np (size) python MC_sim.py
```

MC_sim.py

```
#!/usr/bin/env python
# coding: utf-8
from math import *
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn import linear_model
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure, show



"""
Function that calculates the autocorrelation function for the magnetisation m.

Argument: m -- magnetisation
"""
def fom(m):
    t = np.arange(len(m))
    t_max = t[-1]
    out = []
    for i in t[:-1]:
        sl1 = slice(0,t_max-i)
        sl2 = slice(i,t_max)
        o = 1./(t_max-i)*np.sum(m[sl1]*m[sl2]) -
    (1./(t_max-i))**2*np.sum(m[sl1])*np.sum(m[sl2])
        out.append(o)
    return out
"""
Function that is used for the exponential approximation, after finding tau by integration
    method.

Argument: tau -- correlation time
          t -- time
"""
def f(tau,t):
    return -t/tau

"""
```

```
Function that computes the mean of the susceptibility, using the blocking method. The error
    is computed
after the plot is made.

Argument: T -- Temperature
          M -- total magnetisation of the lattice
          blocknumber -- number of independent blocks

"""

def suscept(T,M,blocknum=5):
    kB = 1
    beta = 1./(kB*T)
    length = len(M)
    mean = np.zeros(blocknum)

    for i in range(blocknum):
        M_arr = M[i*len(M) // blocknum: (i+1)*len(M) // blocknum]
        ss = np.mean(M_arr*M_arr)-np.mean(M_arr)**2
        mean[i] = beta*(ss)/(50*50)

    return mean
"""
Function that computes the mean of the heat capacity, using the blocking method. The error is
    computed
after the plot is made.

Argument: T -- Temperature
          E -- total energy of the lattice
          blocknumber -- number of independent blocks

"""
def cap(T,E,blocknum=5):
    kB = 1
    length = len(E)
    mean = np.zeros(blocknum)

    for i in range(blocknum):
        E_arr = E[i*len(E) // blocknum: (i+1)*len(E) // blocknum]
        ss = np.mean(E_arr*E_arr)-np.mean(E_arr)**2
        mean[i] = ss/(50*50*kB*T**2)

    return mean

#sequence of temperature, defined in the range [1,4], with a step of 0.2
Temperature = np.arange(1,4.2,.2)
"""
Below gives the main properties to be computed (with errors):

-- autocorrelation time (Tval);
-- magnetisation per spin (Mval);
-- Susceptibility per spin (Sval);
-- Heat capacity per spin (Cval);

"""
#sequence of tau values at differnt temperatures, to be computed
Tval = np.zeros(len(Temperature))
#the errors of tau
Terr = np.zeros(len(Temperature))
#sequence of magnetisation per spin values, at different temperatues
Mval = np.zeros(len(Temperature))
#corresponding errors of magnetisation per spin
Merr = np.zeros(len(Temperature))
#sequence of susceptibility per spin values, at different temperatures
Sval = np.zeros(len(Temperature))
#corresponding errors of  susceptibility
Serr = np.zeros(len(Temperature))
#sequence of heat capacity values, at different temperatues
Cval = np.zeros(len(Temperature))
#corresponding errors of heat capacity
Cerr = np.zeros(len(Temperature))

```

```python
for ct,T in enumerate(Temperature):
    print("T=",np.round(T,2))
    fig = figure(figsize=(24,16))
    ax1 = fig.add_subplot(231)
    ax2 = fig.add_subplot(232)
    ax3 = fig.add_subplot(233)
    ax4 = fig.add_subplot(234)
    ax5 = fig.add_subplot(235)
    ax6 = fig.add_subplot(236)
    #number of simulations per temperature
    sim = 8
    #maximum timestep
    tmax = 1000
    M_stack = []
    E_stack = []
    for i in range(sim):
        #for T > 2, we tmax = 10000.
        if ct in np.arange(5,16,1):
            dat =
    np.fromfile('/data/dell5/userdir/ucas_students/ZQ/MC_sim1/T=%.2f'%T+'_sim%d'%(i+1)+'.bin',dtype='float6
            tmax = 10000
            E = dat.reshape(tmax,2)[:,0]
            M = dat.reshape(tmax,2)[:,1]
        else:
            dat =
    np.fromfile('/data/dell5/userdir/ucas_students/ZQ/MC_sim/T=%.2f'%T+'_sim%d'%(i+1)+'.bin',dtype='float6
        #the factor 0.5 is there because the Hamiltonian was summed twice for each timestep
        #reading out total energy E and total magnetisation M, per timestep, the values are
        #appended to two 2d arrays, with axis 0 denoting the timestep, and axis 1 denoting
    the simulation
        E = .5*dat.reshape(tmax,2)[:,0]
        M = dat.reshape(tmax,2)[:,1]
        N = 50
        ax1.plot(E/(N*N))
        ax2.plot(M/(N*N),label='i='+str(i))
        ax1.set_xlabel(r'$t$')
        ax1.set_ylabel(r'$Energy \ per \ spin$')
        ax2.set_xlabel(r'$t$')
        ax2.set_ylabel(r'$Magnetisation \ per \ spin$')

        M_stack.append(M/(N*N))
        E_stack.append(E)

    M_stack = np.asarray(M_stack)
    E_stack = np.asarray(E_stack)
    #using the standard deviation of last 100 timesteps to find a moment when the system is
    equilibrated,
    #which is given by ind_equi
    std_mean = np.mean(np.std(M_stack,axis=0)[-100:])
    std_std = np.std(np.std(M_stack,axis=0)[-100:])
    ind_equi = np.argmin(np.abs(np.std(M_stack,axis=0)-std_mean)>std_std)
    #this grey area depicts the region used for computing the correlation time

    ax2.fill_between(np.arange(tmax)[ind_equi:],y1=M_stack.min(),y2=M_stack.max(),color='silver',alpha=.5)

    #sequence of tau values for all simulations, at current temperature
    ss = np.zeros(sim)
    #list created to append values of heat capacity for all simulations, at current
    temperature T
    HeatCapacity = []
    #list created to append all values of susceptibility for all simulations, at current
    temperature T
    Susceptibility = []
    #list created to append all values of magnetisation per spin, at current temperature T
    Magnetisation = []
    for i in range(sim):
        #stepsize in time sequence
        ds = 1
        #chi values computed using the formula in the 2nd script
        chi = fom(M_stack[i][ind_equi:]/(N*N))
        chi = np.asarray(chi)
        #sequence of time, defined according to the length of the chi sequence
        x_corr = np.arange(0,ds*len(chi),ds)
        #check if all values are greater than 0
        if np.all(chi>=0):
            ind = len(chi)
        #if not, we define an index 'ind', at which we stop the integration to find the tau value
        else:
            ind = np.where(chi<0)[0][0]

        tau = np.sum(chi[:ind]/chi[0]*ds)
        #arrays used to plot the exponential approximation function
        xnew = np.linspace(x_corr[:-1][:ind].min(),x_corr[:-1][:ind].max(),100)
        ynew = f(tau,xnew)
        #plotting the computed chi/chi(0) vs. timestep, together with the approximation function
        ax3.plot(x_corr[:-1][:ind],np.log(chi[:ind]/chi[0]),marker='o')
        ax3.plot(xnew,ynew,ls='--',lw=2,alpha=.4)
        #append the current tau value to the array
        ss[i] = tau
        #the blocks are created to find the means and standard deviations of magnetisation per spin
        blocksize = 16*np.ceil(ss[i]).astype(int)
        #default value of number of blocks is set to be 5
        blocknum = 5

        #if the time period from the equilibration moment to tmax is not long enough to
        #create 5 independent blocks, we should reduce the block number
        while blocknum*blocksize>(tmax-ind_equi+1):

            blocknum-=1
            #blocksize*=.8
            #blocksize=int(blocksize)

        if blocknum>=1:
            df1 = []
            df2 = []
            df3 = []
            for j in range(blocknum):

                df1.append(M_stack[i][ind_equi+blocksize*j:ind_equi+blocksize*(j+1)])
                df2.append((j+1)*np.ones(blocksize))
                df3.append(E_stack[i][ind_equi+blocksize*j:ind_equi+blocksize*(j+1)])

            df1 = np.asarray(df1)
            df2 = np.asarray(df2)
            df3 = np.asarray(df3)
            df1 = df1.flatten()
            df2 = df2.flatten()
            df3 = df3.flatten()

            HeatCapacity.append(cap(T,df3,blocknum))
            Susceptibility.append(suscept(T,df1*(N*N),blocknum))

            dat = {'Magnetisation': df1, 'Group': df2}
            dframe = pd.DataFrame(data=dat)

            #plot the blocks for the last simulation
            if i == 7:
                ax4 = sns.boxplot( dframe['Group'], dframe['Magnetisation'], palette="Pastel2",ax=ax4)
                ax4.set_ylim(df1.min()-.001,df1.max()+.001)
                ax4.set_title('Magnetisation per spin')

            #mean value of magnetisation per spin found for the current temperature
            aa = np.array(dframe.groupby("Group").Magnetisation.mean())
            Magnetisation.append(aa)






        print("The mean and std for tau are: ",ss.mean(),ss.std())
        HeatCapacity = np.asarray(HeatCapacity)
        Susceptibility = np.asarray(Susceptibility)
        Magnetisation = np.asarray(Magnetisation)
        HeatCapacity = np.concatenate(HeatCapacity,axis=None)
        Susceptibility = np.concatenate(Susceptibility ,axis=None)
        Magnetisation = np.concatenate(Magnetisation ,axis=None)

        dat1 = {'Susceptibility': Susceptibility , 'Group': np.ones(len(Susceptibility))}
        dframe1 = pd.DataFrame(data=dat1)

        dat2 = {'Capacity': HeatCapacity, 'Group': np.ones(len(HeatCapacity))}
        dframe2 = pd.DataFrame(data=dat2)


        Sval[ct] = np.array(dframe1.groupby("Group").Susceptibility.mean())[0]
        Serr[ct] = np.array(dframe1.groupby("Group").Susceptibility.std())[0]
        Cval[ct] = np.array(dframe2.groupby("Group").Capacity.mean())[0]
        Cerr[ct] = np.array(dframe2.groupby("Group").Capacity.std())[0]

        ax3.set_xlabel(r'$t$')
        ax3.set_ylabel(r'$\chi/\chi(0)$')


        #plot the blocks for all simulations
        ax5 = sns.boxplot(dframe1['Group'], dframe1['Susceptibility'], palette="Pastel1",ax=ax5)
        ax5 = sns.swarmplot(dframe1['Group'], dframe1['Susceptibility'],
        color="darkslateblue",ax=ax5)
        ax5.set_title('Susceptibility per spin')


        # plot the blocks for all simulation
        ax6 = sns.boxplot(dframe2['Group'], dframe2['Capacity'], palette="Set3_r",ax=ax6)
        ax6 = sns.swarmplot(dframe2['Group'], dframe2['Capacity'], color="orangered",ax=ax6)
        ax6.set_title('Heat Capacity per spin')

        Tval[ct] = ss.mean()
        Terr[ct] = ss.std()
        Mval[ct] = Magnetisation.mean()
        Merr[ct] = Magnetisation.std()



        show()





#plots for Magnetisation, Susceptibility, Heat Capacity and tau

T = np.arange(1,4.2,.2)
plt.figure(figsize=(7.5,6))
ax = plt.axes()
plt.scatter(T,Mval,marker='o',color='crimson',edgecolor='k',zorder=3)
plt.plot(T,Mval,lw=2,color='darkviolet',zorder=2)
plt.fill_between(T, Mval-Merr, Mval+Merr,color='plum',alpha=.7)
plt.xlabel('Temperature')
plt.ylabel('Magnetisation per spin')
plt.title('Magnetisation')
ax.set_facecolor('lavender')
plt.grid(color='white',zorder=1)
plt.show()

plt.figure(figsize=(7.5,6))
ax = plt.axes()
plt.scatter(T,Sval,marker='o',color='g',edgecolor='k',zorder=3)
plt.plot(T,Sval,lw=2,color='darkslategrey',zorder=2)
plt.fill_between(T, Sval-Serr, Sval+Serr,color='darkkhaki',alpha=.7)
plt.xlabel('Temperature')
plt.ylabel('Susceptibility per spin')
plt.title('Susceptibility')
```

```
ax.set_facecolor('palegoldenrod')
plt.grid(color='white',zorder=1)
plt.show()

plt.figure(figsize=(7.5,6))
ax = plt.axes()
plt.scatter(T,Cval,marker='o',color='darkorange',edgecolor='k',zorder=3)
plt.plot(T,Cval,lw=2,color='darkgoldenrod',zorder=2)
plt.fill_between(T, Cval-Cerr, Cval+Cerr,color='burlywood',alpha=.7)
plt.xlabel('Temperature')
plt.ylabel('Heat Capacity per spin')
plt.title(r'$Heat \ Capacity$')
ax.set_facecolor('navajowhite')
plt.grid(color='white',zorder=1)
plt.show()

plt.figure(figsize=(7.5,6))
ax = plt.axes()
plt.scatter(T,Tval,marker='o',color='blue',edgecolor='k',zorder=3)
plt.plot(T,Tval,lw=2,color='midnightblue',zorder=2)
plt.fill_between(T, Tval-Terr, Tval+Terr,color='cornflowerblue',alpha=.7)
plt.xlabel('Temperature')
plt.ylabel(r'$\tau$')
plt.title('Autocorrelation time')
ax.set_facecolor('powderblue')
plt.grid(color='white',zorder=1)
plt.show()
```

MC_properties.py