Ising model simulation using Metropolis Algorithm

Kartik Chhajed

June 6, 2020

Contents

Abstract

This report aims to give a short description of the Ising two-dimensional model and an introduction to Monte Carlo simulation. To draw results, we implemented the Metropolis-Hastings algorithm, which helps us to built likely configurations at a particular temperature. In order to approximate critical exponents, we did finite-size scaling. At the end of this report, we included optimized python code for doing the simulation.

Introduction

Physicist Wilhelm Lenz first introduced the Ising model in 1920 as a problem to his student Ernst Ising. Ernst Ising solved the one-dimensional Ising model and showed that the model does not show the phase transition [1]. Two decades later, Onsager solved the two-dimensional Ising model using the transfer matrix method [2].Surprisingly, the two-dimensional square-lattice Ising model, which is a simplified model of reality exhibits phase transition. Onsager showed that there is a specific temperature, called the Curie temperature or critical temperature, T_c below which the system shows ferromagnetic long-range order. Above it, it is paramagnetic and is disordered.

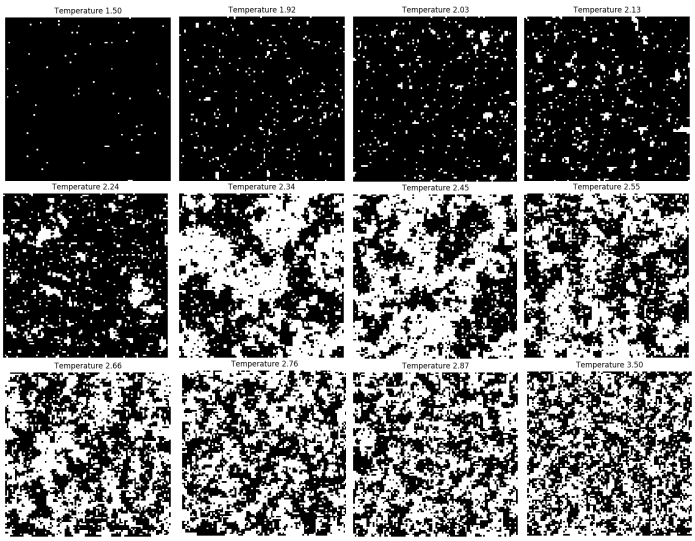


Figure 1: Two-dimensional Ising model simulation on 100x100 lattice. From left to right and top to bottom, the temperature is increasing. At equilibrium, when $T < T_c$, typical configurations in the + phase look like a "sea" of + spins with "islands" of − spins. For larger lattice size, the "island" have "lakes" of + spins. In this picture, + spins are in black, − spins are in white. Each connected white object is a cluster.

At zero temperature, every spin is aligned in either +1 (or -1) direction. When we increase the temperature, keeping below T_c, some spin of starts orienting themselves in the opposite direction. The typical length scale of cluster

forming is called correlation length, ξ, and it grows as we increase the temperature and diverges at T_c. If we go beyond T_c, the correlation length starts decreasing, and at the infinite temperature, it becomes zero fig. 1.

As the correlation length diverges at a critical point, the system becomes blind to the microscopic dynamics of the system. That means, the critical behaviour might not depend on details of the model (underlying lattice, precise range of interaction), it does depend on general features (short-ranged or long-ranged interaction, symmetry group.). It also suggests that all system with the same general features belongs to the same Universality class near criticality [3].

The Widom scaling hypothesizes that near the critical point, the free energy is a homogeneous function of the external field, h, and reduced temperature, t. Widom's hypothesis can be proven by regrouping Ising spin in a larger block; it is known as Block-Spin Renormalization. In such a transformation, the Hamiltonian is invariant [4]. Block-Spin Renormalization tells that all the six critical exponents which describe the behaviour of physical quantities near-continuous phase transitions can be expressed in terms of any two critical exponents.

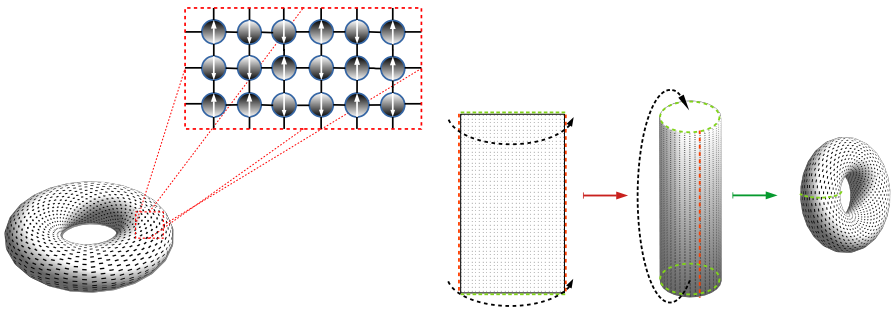
Chapter 1

Classical Ising Model

The Ising model is a statistical model of spins, $S_i = \pm 1$, described on a lattice of dimension, d. The spins can interact with external field and nearest neighbour. Hamiltonian for such a system can be written as

$$\mathcal{H} = -J \sum_{\langle ij \rangle} S_i S_j - H \sum_i S_i$$

where $\langle ij \rangle$ means that we are summing over nearest neighbours. When $J > 0$, neighboring spins prefer to align is same direction, viz., either $\uparrow\uparrow$ or $\downarrow\downarrow$. In the context of magnetism, such a system is called a *ferromagnet.* When $J < 0$, neighbouring spins prefer to align is the opposite direction, viz., $\uparrow\downarrow$, and such a system is called a *anti-ferromagnet.* For the rest of the discussion, we will focus on two-dimensional ferromagnetic system (fig. 1.1a).



(a) Ising model on torus. (b) Sheet folded into a torus.

Figure 1.1: Two-dimensional Ising model with periodic boundary condition.

We will make spins at edges of the lattice to interact with geometrical opposite edges. This is known as the periodic boundary condition. It can be visualized more clearly if we consider a two-dimensional rubber sheet being folded into a torus with spins being on the surface of this topological structure (fig. 1.1b).

The partition function, $\mathcal{Z}$, and free energy, $\mathcal{F}$, are given as

$$\mathcal{Z} = \mathrm{Tr} e^{-\beta \mathcal{H}}, \ \mathcal{F} = -\frac{\ln \mathcal{Z}}{\beta} \tag{1.1}$$

The partition function cannot be calculated analytically for general lattices, in arbitrary dimension d. Exact solution for ising model is known for $d = 1$ and, when external field is zero, in $d = 2$. For arbitrary dimension $d > 2$ and, for $d = 2$ in presence of external field are not determined exactly. The first successful attempt to find the exact solution for dimension $d = 2$, in the absence of external field, was done by Onsager is famously complicated [2]. Since then, simpler solutions using more modern techniques have been discovered [5].

1.1 Phase transition and Critical Phenomena

Important thermodynamically measurable quantities for the Ising model is magnetization or the magnetic moment per site, M:

$$M = \frac{1}{N}\sum_i \langle S_i \rangle = -\frac{1}{N}\frac{\partial \mathcal{F}}{\partial H} \tag{1.2}$$

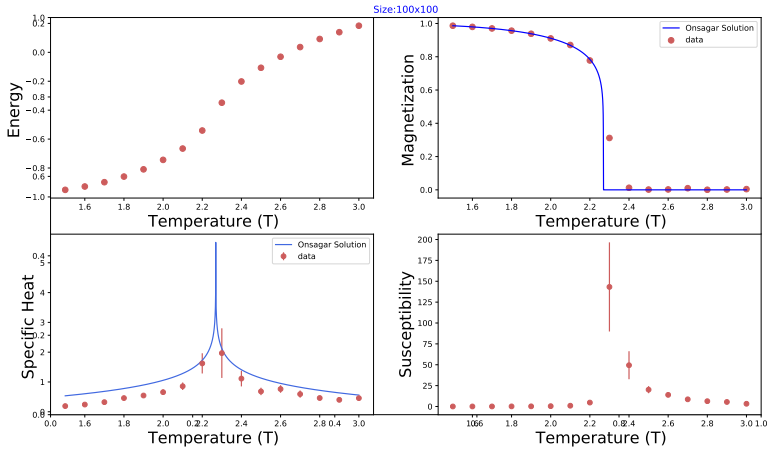


Figure 1.2: Two-dimensional Ising model simulation on 100x100 lattice. For T $\gtrsim$ 2.269 it shows the system is in symmetric paramagnetic state with vanishing spontaneous magnetization, $m \sim 0$. While for T $\lesssim$ 2.269, it shows that the system is an ordered ferromagnetic phase, $m \neq 0$. The x-axis is dimensionless temperature, T $\equiv k_B T/J$.

In absence external field, when we plot magnetization as a function of temperature (fig. 1.2), we see below a certain temperature, T_c, the magnetization[1]

$$M \sim (T_c - T)^{\beta}; \qquad \beta = 1/8 \tag{1.3}$$

And above T_c, the magnetization is zero. This could be simply understood as energy-entropy competition. $J > 0$ will make energetically favourable for all spin to align in the same direction (ordered phase), whereas the effect of

[1]Here, the critical exponent β is not to be confused with the inverse temperature. To avoid confusion, we will only refer β as critical exponent till the end of this section.

temperature will be to randomize the spins (disordered phase), with entropy winning out over energy.

Another essential quantity of the Ising model is the magnetic susceptibility, χ:

$$\chi = \frac{\partial M}{\partial H}$$

The susceptibility show divergent behavior near critical temperature (fig. 1.2) according to power law

$$\chi \sim |T - T_c|^{-\gamma}; \qquad \gamma = \frac{7}{4} \tag{1.4}$$

Let's construct a general expression for the susceptibility

$$\begin{aligned}
\chi &= \frac{\partial M}{\partial H} \\
&= \frac{kT}{N} \frac{\partial^2 \ln \mathcal{Z}}{\partial H^2} \\
&= \frac{kT}{N} \Big[\frac{1}{\mathcal{Z}} \frac{\partial^2 \mathcal{Z}}{\partial H^2} - \frac{1}{\mathcal{Z}^2} \Big(\frac{\partial \mathcal{Z}}{\partial H} \Big)^2 \Big] \\
&= \frac{1}{NkT} \Big[\sum_{i,j} \langle S_i S_j \rangle - \Big(\sum_i \langle S_i \rangle \Big)^2 \Big] \\
&= \frac{1}{NkT} \sum_{i,j} \Gamma(i-j)
\end{aligned}$$

Above equation known as *static susceptibility sum rule.* Here $\Gamma(i-j)$ is two-point correlation function defined as

$$\Gamma(i-j) = \langle S_i S_j \rangle - \langle S_i \rangle \langle S_j \rangle \tag{1.5}$$

When system has translational symmetry we can write the magnetic susceptibility as

$$\chi = \beta \sum_i \Gamma(i)$$

The correlation length, $\xi(T)$ is characteristic length at which the value of correlation function $\Gamma(i)$ has decayed to e^{-1}:

$$\Gamma(i) \sim \exp\left(\frac{|i|}{\xi(T)} \right)$$

And

$$\xi(T) \sim |T - T_c|^{-\nu}; \qquad \nu = 1 \tag{1.6}$$

Diverging correlation exceeds the physical length of the system near criticality and due to 'lack of room', and correlation decays according to power law:

$$\Gamma(i) \sim \frac{1}{|n|^{d-2+\eta}} \tag{1.7}$$

and for two-dimensional ising model

$$\Gamma(i) \sim \frac{1}{|n|^{\eta}}; \qquad \eta = \frac{1}{4}$$

All exponent relations eq. (1.3), eq. (1.4), eq. (1.6), eq. (1.7) along with two more defined in table 1.1 are called critical exponent. Critical Exponent fall into universality class and obey the scaling relations as

$$\nu d = 2 - \alpha = 2\beta + \gamma = \beta(\delta + 1) = \gamma\frac{\delta + 1}{\delta - 1}$$

$$2 - \eta = \frac{\gamma}{\nu} = d\frac{\delta - 1}{\delta + 1}$$

These scaling relations imply that there are only two independent exponents, e.g., ν and η. All this follows from the renormalization group theory.

Table 1.1: Critical exponent relation and to the right are the analytical value calculated by Onsager in 1944 for 2-dimensional Ising model[2].

Critical Exponent	Definition	Ising Value
α	$C \propto (T - T_c)^{-\alpha}$	0
β	$M \propto (T - T_c)^{\beta}$	1/8
γ	$\chi \propto (T - T_c)^{-\gamma}$	7/4
δ	$M \propto h^{1/\delta}$	15
ν	$\xi \propto (T - T_c)^{-\nu}$	1
η	$\Gamma(n) \propto \lvert n\rvert^{2-d-\eta}$	1/4

1.2 Exact Solutions

Using Kramers–Wannier duality relation [6], we can relate the partition function of a two-dimensional square-lattice Ising model at a low temperature to that of same Ising model at a high temperature to get the critical temperature as a relation:

$$\sinh\left(\frac{2J}{k_B T_c}\right) = 1 \implies k_B T_c = \frac{2J}{\ln(1 + \sqrt{2})} \backsimeq 2.269J \tag{1.8}$$

The Onsager's formula for absolute magnetization per spin (m) is given as

$$m = \begin{cases} \left(1 - \left(\sinh 2\beta J\right)^{-4}\right)^{1/8} & T < T_c \\ 0 & T > T_c \end{cases} \tag{1.9}$$

The behavior of specific heat, C near T_c is given by

$$C \approx -Nk\frac{2}{\pi}\left(\frac{2J}{kT_c}\right)^2 \ln\left|1 - \frac{T}{T_c}\right| \qquad (T \text{ near } T_c)$$

Chapter 2

Numerical simulation and Numerical tools

In this chapter, we will discuss the numerical method and numerical tools required to do the simulation of the Ising model. We also provide tips to do efficient simulation saving time. In the first section, we give an algorithm to do the simulation. In subsequent sections, we discuss the method to measure quantities like magnetization, specific heat. In the last section, we discuss the approach to estimate the critical exponent.

2.1 Monte Carlo Algorithm

We can start with any arbitrary initial configuration of the lattice. However, we can smartly choose a "good"[1] initial condition so that system reaches equilibrium early. As long as our system is ergodic, we are allowed to do so.

Now, we thermalize our system by performing Monte Carlo steps for a large number of times. In one Monte Carlo step, we flip one spin at a time. This is herein as "Single Flip Spin Dynamics". This flipped spin is accepted or rejected with the probabilities depending on how it affects the change in energy of the system. The algorithm for identifying and performing this step is given below:

1. Select any one spin at random.
2. Flip that spin.
3. Calculate the change in energy to its previous state.
4. Decide whether to accept the flipped spin or not.
5. Repeat until the desired configuration meets.

In our simulation, we will start with all spin up and thermalize it for $T = \min(\{T_i\})$. The equilibrium configuration can be used as the initial state for successive temperature points.

[1] By good initial point we mean an arrangement which is close to an equilibrium configuration.

In Monte Carlo steps, the decision is made by the Metropolis-Hastings Algorithm. This sampling technique uses the idea of Markov chain of successive configurations $\{s_i\}$ where each configuration $\{s_j\}$ is constructed from a previous configuration $\{s_i\}$ via a suitable transition probability $W(\{s_i\} \to \{s_j\})$. The main idea is that, when reaching equilibrium, it should satisfy the detailed balance condition. i.e. there is an equal probability for the transitions $\{s_i\} \to \{s_j\}$ and $\{s_j\} \to \{s_i\}$. The transition probability is given by

$$W(\{s_i\} \to \{s_j\}) = \begin{cases} e^{-\beta \Delta E} & \text{if } \Delta E > 0 \\ 1 & \text{otherwise} \end{cases}$$

The algorithm for making decision whether or not to accept the flip spin is given below.

1. If $\Delta E < 0$, accept the flipped spin.
2. If not then, only accept with probability $e^{-\beta \Delta E}$.[2]
3. If both step 1 and 2 fail, return the flipped spin into its previous state.

2.2 Calculation of observables

Calculation of observables is straight forward. The magnetization would be given by time averaging of magnetizations:

$$m = \langle m^\alpha \rangle = \left\langle \frac{1}{N} \sum_i s_i^\alpha \right\rangle$$

Here α index corresponds to configurations in time. The variance of magnetization times inverse temperature will give Susceptibility.

$$\chi = \beta N (\langle m^2 \rangle - \langle m \rangle^2)$$

Similarly, energy is given by time averaging of energies: $E = \langle E^\alpha \rangle$. The specific heat is given by variance of energies times inverse temperature square, viz.,

$$c = \frac{k\beta^2}{N} (\langle E^2 \rangle - \langle E \rangle^2)$$

As said, the calculation is straight forward, but the question is when we should start measuring and at what intervals should we measure? We can start measuring when the system reaches equilibrium, and we can measure at intervals 2τ, where τ is correlation time.

The correlation-time can be calculated by fitting autocorrelation function $\chi(t)$ in exponential.

$$\chi(t) = \frac{1}{t_{max} - t} \sum_{t'=0}^{t_{max}-t} m(t') m(t'+t) - \frac{1}{t_{max} - t} \sum_{t'=0}^{t_{max}-t} m(t') \frac{1}{t_{max} - t} \sum_{t'=0}^{t_{max}-t} m(t'+t) \sim e^{-t/\tau}$$

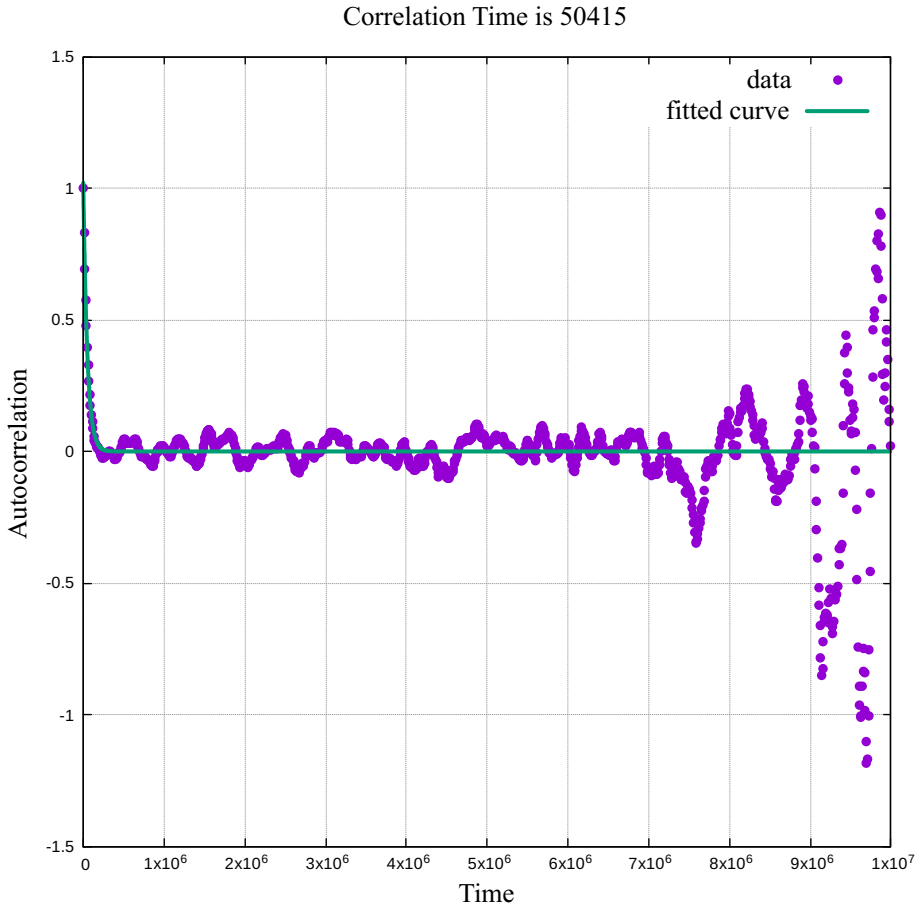


Figure 2.1: Autocorrelation function fitted with exponential for temperature $T = 3.5J/k_B$. The initial configuration was all spin up. Here time is Monte Carlo steps. After autocorrelation function becomes zero, it starts showing oscillating fluctuation. This suggests that we can fit curve for $\chi(t) > 0$ and the latter part can be discarded. This will save computational time.

To make fitting more efficient, we can fit by taking logarithm of autocorrelation function with weight, $w = \sqrt{\chi(t)}$ and fit it with linear function $y(x) = Ax + B$. We will be intrested in value of A. Since we are fitting $\chi(t) > 0$ part, we do not need to worry about non-positive value inside the $\ln(\dots)$.

Depending on the number of measurements, the error is calculated using the Jackknife method or Bootstrap method. In our analysis, the error analysis of specific heat and Suceptibiliy is done using Jack-Knife method.

Algorithm

1. Take arrays of Energy for particular temperature. Divide the set of energies into $n = \dfrac{t_{MAX}}{2\tau}$, where t_{MAX} is total number of Monte Carlo steps.

[2]To make our code more efficient we can store exponential value as an array.

2. Calculate the variance (which we call σ_n) and multiplying it with β^2 which gives specific heat for that temperature.

3. Now, we use the Jackknife method: discarding i^{th} measurement, we calculate variance, σ_i.

4. Now comes the error estimation, which is given by β^2 times square root of the Jackknfe varience viz.

$$\sigma_{JK} = \sqrt{\sum_{i=1}^{N} (\sigma_n - \sigma_i)^2}$$

2.3 Finite-size scaling

Finite-size scaling method determines the critical control parameter of the phase transition and extracts critical exponents by observing how physical quantities vary with the system size as well as the control parameter. The size of the system becomes significant when the correlation length of the system is comparable to the length scale of the system size.

$$\xi \sim L \tag{2.1}$$

In such a regime, the correlation length of the finite system cannot diverge, hence cannot exhibit complete critical phenomena. Susceptibility, which is proportional to a power of correlation length, also does not diverge (Which is evident because it is expressed in a finite sum). A dimensionless function, χ_0

$$\chi \sim \xi^{\gamma/\nu} \tag{2.2}$$

$$\chi = \xi^{\gamma/\nu} \chi_0(L/\xi) \tag{2.3}$$

χ_0 has following asymptotic behavior.

$$\chi_0(x) = \begin{cases} 0 & (x \gg 1) \\ \sim x^{\gamma/\nu} & (x \ll 1) \end{cases} \tag{2.4}$$

Chapter 3

Results and Conclusion

3.1 Phase transition

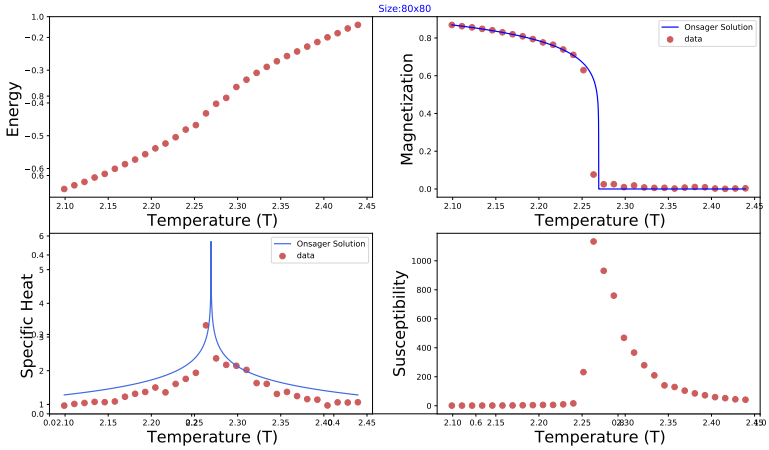
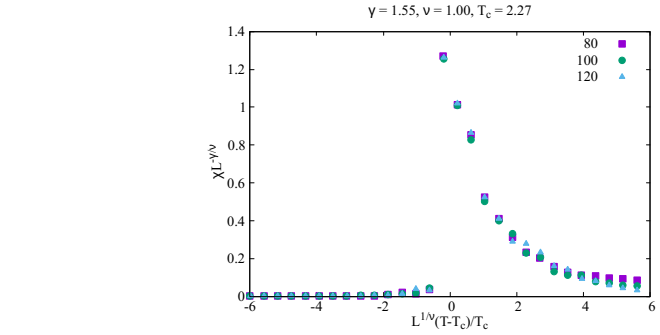
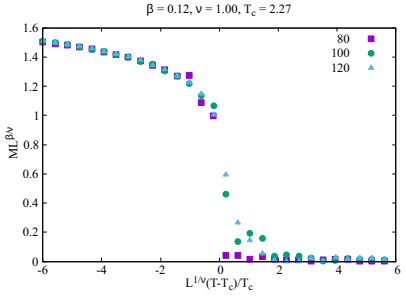


Figure 3.1: Results of the two-dimensional Ising model simulation on 80x80 lattice. For T $\gtrsim$ 2.269 it shows the system is in a symmetric paramagnetic state with vanishing spontaneous magnetization, $m \sim 0$. While for T $\lesssim$ 2.269, it shows that the system is an ordered ferromagnetic phase, $m \neq 0$. The x-axis is dimensionless, T $\equiv k_B T/J$.

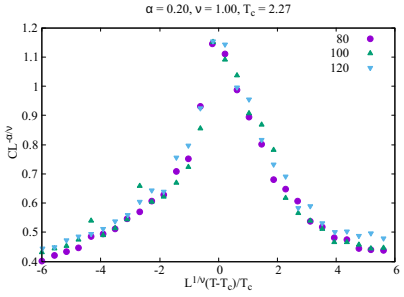
3.2 Finite size scaling



(a) Suceptibiliy scaling function.



(b) Magnetization scaling function.



(c) Specific heat scaling function..

Figure 3.2: Data collapse of a) Suceptibiliy, b) Magnetization, and c) Specific heat for two dimensional ising model for three different sizes. From data collapse we find $\alpha = 0.02,\ \beta = 0.12,\ \gamma = 1.55,$ and $\nu = 1.00$. The temperature range chosen to be inversely proportional to system size.

Table 3.1: Comparison between critical exponent obtained from simulation and expected values from the literature.

Critical Exponent	Definition	Expected Value	Value from simulation
α	$C \propto (T - T_c)^{-\alpha}$	0	0.20
β	$M \propto (T - T_c)^{\beta}$	1/8	0.12
γ	$\chi \propto (T - T_c)^{-\gamma}$	7/4	1.55

3.3 Problems and Comments

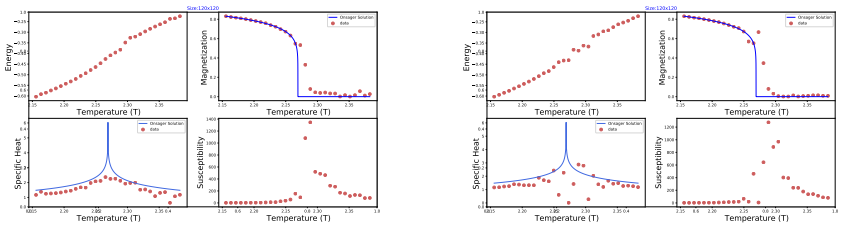
We end this report with an informal discussion regarding performance and generating results.

1. A good choice of pseudo-random number is MT19937. The Random library of Python uses MT19937. However, NumPy's Generator uses bits provided by PCG64 which has better statistical properties than the legacy MT19937 used in RandomState. We used both these in our code.

2. It is good to use NumPy library when dealing with the array. Its backend runs using C/C++/Fortran which are better in performance. In our code, we used decorator Numba which makes convert code to Low-level language. However, the decoration can be applied to a few function only.

3. Python by default single thread. If we care less about our machine and want high performance, Numba provides an option for multi-thread processing.

4. The Binder cumulant provides a more accurate technique to extract the critical temperature.



(a) 10^5 Monte Carlo steps. (b) 10^6 Monte Carlo steps.

Figure 3.3: First, I thought by increasing the number of Monte Carlo steps we can get better results. However, for system size 120×120, this turns out to be another way around for specific heat.

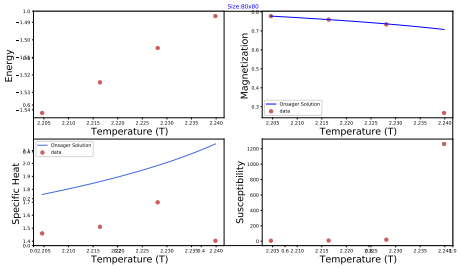


Figure 3.4: Because of critical slowing down, we performed few more Monte Carlo steps where results were not close to Onsager solutions. The condition for doing more simulation near the critical point was later included in the code.

Bibliography

[1] Ernst Ising. *Beitrag zur Theorie des Ferromagnetismus*. PhD thesis, February 1925.

[2] Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Phys. Rev.*, 65:117–149, Feb 1944.

[3] Leo P. Kadanoff. Scaling laws for ising models near T_c. *Physics Physique Fizika*, 2:263–272, Jun 1966.

[4] Kerson Huang. *Statistical Mechanics*. John Wiley and Sons, 1987.

[5] R J Baxter and I G Enting. 399th solution of the ising model. *Journal of Physics A: Mathematical and General*, 11(12):2463–2473, dec 1978.

[6] H. A. Kramers and G. H. Wannier. Statistics of the two-dimensional ferromagnet. part i. *Phys. Rev.*, 60:252–262, Aug 1941.

Appendix A

Python code for Ising Model simulation

```
from numba import jit
import numpy as np
from random import random
import matplotlib.pyplot as plt
import time
from tqdm import trange
import logging

logging.basicConfig(level=logging.INFO,filename='simulation.log',
    filemode='w',format='%(asctime)s - %(message)s',datefmt='%d-%b
    -%y %H:%M:%S')
np.seterr(all='warn')

################################################
#                                              #
#              SIMULATION MACROS               #
#                                              #
################################################
show_interactive_plots = "Off"
if show_interactive_plots == "On":
    plt.ion() #interactive plots ON
"""----------------------------------------
Simulation MACROs:
T_max and T_min is range of temperature.
nt is number of Temperature points.
sweeps are number of mc steps per spin.
min_meas is minimum number Measurement.
j_knife_factor is jack knife factor is used when number of
    measurement interval < 2 x Correlation time.
All some_variables0 are default value.
----------------------------------------"""
logging.info("Starting Ising Model Simulation")
T_min = 1.5; T_max = 3
nt = int((T_max-T_min)*10+1)
sweeps0 = 1000000
max_sweeps = sweeps0*10
min_meas = 100
j_knife_factor0 = 1
startTime = time.time()
T = np.linspace(T_min, T_max, nt)
```

```
"""
We will work with expanding lattices. We will store expanded
    lattice for particular temperature. Stored lattice would be
    used as initial configuration for higher dimenssion lattic size
    . We have two methods for expanding lattice: zooming and
    stacking. We recommend stacking for use.
"""
states = {_: None for _ in T}
#lattice_sizes = 3**(np.arange(2,5))
#################OR##################
lattice_sizes = 2**(np.arange(4,8))

################################################
#                                              #
#                   FUNCTIONS                  #
#                                              #
################################################
"""Onsagar's solutions"""
def onsagar_specific_heat(X):
    const = -(2/2.269)**2*2/np.pi
    return const*np.log(abs(np.ones(len(X))-X/2.269))
@jit(nopython=True)
def onsagar_mag(X):
    lst1 = (1-(np.sinh(np.log(1+np.sqrt(2))*2.269/X[X<2.269]))
    **(-4))**(1/8)
    lst2 = 0*X[X>=2.269]
    return np.concatenate((lst1,lst2))


"""Monte Carlo Metropolis algorithm"""
@jit(nopython=True, parallel = True)
def monteCarlo(n, state, energy, mag, beta, sweeps,max_sweeps):
    if sweeps > max_sweeps:
        sweeps = max_sweeps
    exp_betas = np.exp(-beta*np.arange(0,9))
    energies, mags = np.zeros(sweeps), np.zeros(sweeps)
    # random state indices
    J = np.random.randint(0, n, size=(sweeps, n*n))
    K = np.random.randint(0, n, size=(sweeps, n*n))
    #loop
    for t in range(sweeps):
        for tt in range(n*n):
            # random indices
            j, k = J[t, tt], K[t, tt]
            s = state[j,k]
            neighbour_sum = (state[(j-1)%n, k] +
                             state[j, (k-1)%n] + state[j, (k+1)%n]
    +
                             state[(j+1)%n, k])
            energy_diff = 2*s*neighbour_sum
            if energy_diff < 0 or random() < exp_betas[energy_diff
    ]:
                s *= -1
                energy += energy_diff
                mag += 2*s
            state[j, k] = s
        energies[t], mags[t] = energy, mag
    return energies, mags


"""Calculation of auto-correlation"""
def autocorrelation(M):
```

```
    start_time = time.time()
    tau = 1
    sweeps = len(M)
    auto = np.zeros(sweeps)
    for t in range(sweeps):
        some_time = sweeps-t
        first_term = np.average(M[:some_time]*M[t:sweeps])
        S1 = np.average(M[:some_time])
        S2 = np.average(M[t:sweeps])
        auto_temp = first_term - S1*S2
        if auto_temp > 0:
            auto[t] = auto_temp
        else:#remove oscillating part
            break
    if auto[0] != 0:
        auto = auto[auto>0]
        auto = auto/auto[0] #normalization
        len_auto = len(auto)
        if len_auto > 1: #draw a straight line if you have atleast two points
            tau = int(-1/np.polyfit(np.arange(len_auto), np.log(auto), 1, w=np.sqrt(auto))[0])
    tau = max(tau,1)
    logging.info(f"Correlation time = {tau}")
    return tau


"""
Calculation of specific heat or Susceptibility and errorbar.
CX is Specific Heat or Susceptibility.
CX_i is Specific Heat or Susceptibility without i-th measurement.
"""
@jit(nopython=True, parallel = True)
def jackKnife(EM,factor=1):
    n = len(EM)
    CX = np.var(EM)
    CX_i = np.zeros(n)
    for i in range(n):
        CX_i[i] = np.var(np.delete(EM,i))
    under = np.sum(np.square(np.full(n,CX) - CX_i))
    CX_err = np.sqrt(under*factor)
    return CX, CX_err


"""
Zooming lattice: Number of old bonds will double (z times, where z is zoom factor) in new lattice. If all spins were alighned, energy would have been (z*n)**2. But, 1/z are old bonds. Magnetization would also increase as system size increase as it is a extensive state variables.
"""
def zoomLattice(z,state,energy,mag):
    Izxz = np.ones((z,z),dtype= "int")
    return (np.kron(state,Izxz), energy*z-(z-1)*(z*n)**2//z, z*z*mag)


"""
Stacking Lattices: Stacking z lattice and taking advantage of periodic boundary condition. The energy and magnetization would also increase as system size increase as they are extensive state variables. Other trick to explore is Zoom.
```

```
"""
def stackLattice(z,state,energy,mag):
    h_stack_state = state
    for _ in range(z-1):
        h_stack_state = np.hstack((h_stack_state,state))
    v_stack_state = h_stack_state
    for _ in range(z-1):
        v_stack_state = np.vstack((v_stack_state,h_stack_state))
    return (v_stack_state, z*z*energy, z*z*mag)


#this will show configuration of lattice
def showLattice(lattice):
    n = len(lattice)
    plt.axis('off')
    plt.axis([-0.5, n-0.5, -0.5, n-0.5])
    plt.title(f'Temperature {temp}, Size = {n}x{n}')
    plt.imshow(lattice,cmap=plt.get_cmap('gray'),vmin=-1,vmax=1)
    plt.show()
    plt.pause(0.1)

##################################################
#                                                #
#                     MAIN                       #
#                                                #
##################################################
"""we will plot the following wrt temperature, T"""
plotEnergy = np.zeros(nt)
plotMag = np.zeros(nt)
plotChi = np.zeros(nt)
plotChi_err = np.zeros(nt)
plotSH = np.zeros(nt)
plotSH_err = np.zeros(nt)
plotCorrelation = np.zeros(nt)


"""
Preparing n x n lattice with all spins up.
Here, z is a zoom factor or a stacking factor.
"""
n = min(lattice_sizes)
N = n*n
z = lattice_sizes[1]//lattice_sizes[0]
state = np.ones((n,n),dtype="int")
energy, mag = -N, N
"""lattice size loop"""
for n in lattice_sizes:
    logging.info(f"Lattice size is {n}x{n}")
    print(f"Lattice size is {n}x{n}")
    N = n*n
    """temperature loop"""
    for k in trange(nt):
        temp = T[k]
        Beta=1/temp
        if states[temp] !=  None:
            (state,energy,mag) = states[temp]
        logging.info("_"*35)
        logging.info("Temperature is %0.2f, time elapsed %d" %(temp
    ,time.time()-startTime))
        sweeps = sweeps0; j_knife_factor = j_knife_factor0;
    measurements = 0
        E, M = np.zeros(0), np.zeros(0)
```

```
        while measurements < min_meas:
            energies, mags = monteCarlo(n, state, energy, mag, Beta
    , sweeps, max_sweeps//10)
            energy, mag = energies[-1], mags[-1]
            E = np.concatenate((E,energies))
            M = np.concatenate((M,mags))
            delta_int = eq_time = 2*autocorrelation(M)
            measurements = len(E[eq_time::delta_int])
            logging.info(f"{measurements} measurements are possible
    ")
            if  measurements < min_meas:
                _energies_ = len(E)
                if _energies_ < max_sweeps:
                    sweeps = delta_int*(min_meas-measurements)
                    logging.info(f"\tdoing {sweeps} more sweeps")
                else:
                    delta_int = (_energies_-eq_time)//min_meas
                    j_knife_factor = eq_time/delta_int
                    measurements = len(E[eq_time::delta_int])
                    logging.info(f"We will do {measurements}
    measurements")
        if show_interactive_plots == "On":
            showLattice(state)


        #doing measurements
        E = E[eq_time::delta_int]
        M = M[eq_time::delta_int]
        plotMag[k] = np.average(M)/N
        Chi, Chi_err = jackKnife(M,j_knife_factor)
        plotChi[k] =Chi*Beta/N
        plotChi_err[k] =Chi_err*Beta/N
        plotEnergy[k] = np.average(E)/N
        sp_heat, sp_heat_err = jackKnife(E,j_knife_factor)
        plotSH[k] = sp_heat*Beta*Beta/N
        plotSH_err[k] = sp_heat_err*Beta*Beta/N
        plotCorrelation[k] = eq_time//2


        #lattice expansion
        states[temp] = stackLattice(z,state,energy,mag)
        #states[temp] = zoomLattice(z,state,energy,mag)


    #PLOTS##PLOTS##PLOTS##PLOTS##PLOTS##PLOTS##PLOTS##PLOTS#
    f = plt.figure(figsize=(16, 9));
    title_name = "Size:"+str(n)+"x"+str(n)
    plt.title(title_name, color='b');

    sp =  f.add_subplot(2, 2, 1 );
    plt.scatter(T, plotEnergy, s=50, marker='o', color='IndianRed')
    plt.xlabel("Temperature (T)", fontsize=20);
    plt.ylabel("Energy ", fontsize=20); plt.axis('tight');

    sp =  f.add_subplot(2, 2, 2 );
    plt.scatter(T, abs(np.array(plotMag)), s=50, marker='o', color=
    'IndianRed', label = "data")
    temp_list = np.linspace(T_min, T_max, 10000)
    plt.plot(temp_list, onsagar_mag(temp_list) , color='blue',
    label = "Onsager Solution")
    plt.legend()
    plt.xlabel("Temperature (T)", fontsize=20);
```

```
    plt.ylabel("Magnetization ", fontsize=20);   plt.axis('tight');

    sp =  f.add_subplot(2, 2, 3 );
    plt.errorbar(T, plotSH, yerr = plotSH_err, fmt='o', color='
    IndianRed', label = "data")
    plt.plot(temp_list, onsagar_specific_heat(temp_list), color='
    RoyalBlue', label = "Onsager Solution")
    plt.legend()
    plt.xlabel("Temperature (T)", fontsize=20);
    plt.ylabel("Specific Heat ", fontsize=20);   plt.axis('tight');

    sp =  f.add_subplot(2, 2, 4 );
    plt.errorbar(T, plotChi, yerr = plotChi_err, fmt='o', color='
    IndianRed', label = "data")
    plt.xlabel("Temperature (T)", fontsize=20);
    plt.ylabel("Susceptibility", fontsize=20);   plt.axis('tight');

    #timeIs = time.strftime("%H-%M-%S")
    #plt.savefig(timeIs+'.pdf')
    plt.show()

    #storing measurements in in a file
    with open(str(n)+"data","w") as file:
        file.write("##Temp\tEnergy\tMag\tSp_ht\tSp_ht_err\tChi\
    tChi_err\ttau\n")
        for i in range(nt):
            file.write(str(T[i])+"\t"+str(plotEnergy[i])+"\t"+str(
    plotMag[i])+"\t"+str(plotSH[i])+"\t"+str(plotSH_err[i])+"\t"+
    str(plotChi[i])+"\t"+str(plotChi_err[i])+"\t"+str(
    plotCorrelation[i])+"\t"+"\n")
```