Analytical and Numerical Solution in Ising Model

전성배 2021550026

서울시립대학교 물리학과

2022 하계고체인턴에서 Ising Model을 연구하였다. Ising Model을 Analytical Solution과 Numerical Solution으로 접근 방식으로 나누어서 Ising Model을 탐구하였다. Analytical Solution을 구하기 위해, Ising의 One-dimensional Ising Model의 Partition Function 유도 과정을 따라가보며 Energy, Heat Capacity, Magnetization, Susceptibility의 Exact Solution을 구하였다. Numerical Solution을 구하기 위해 먼저 Solution을 구하는데 필요한 개념들과 Markov Chain Monte Carlo (MCMC), Metropolis Algorithm을 학습하였다. 이를 이용하여 One-dimensional Ising Model의 Numerical Solution을 구해보았다. 이 두 가지 접근 방법에 따라 구한 해를 비교하며 Numerical Solution의 정확도에 대해 탐구해보았다.
동일한 방법으로, Two-dimensional Ising model으로 구현하여 Numerical Solution을 구하였고, One-dimensional Ising Model에서 설명하지 못하였던 상전이가 Two-dimensional Ising model에서는 나타남을 알고 상전이가 이루어지는 Critical temperature를 확인하였다.

Key word: Ising Model, Markov Chain, Monte Carlo, Metropolis Algorithm, Detailed balance

이 보고서는 Analytical Solution의 증명과 증명에 필요한 개념들을 서술하고, 이후에 Numerical Solution의 내용을 담고자 한다. Numerical Solution을 구한 후, Analytical Solution을 구한 여름고체인턴의 진행 과정과 반대이지만 실제로 탐구 순서가 중요한 것이 아닌, 각각의 개념과 그 개념의 유기적인 연결을 위주로 보고서를 작성하는 것이 더 좋다고 판단하였다. 따라서 순서를 변경하여 Analytical Solution에 대해 먼저 서술하고자 한다.

I. Analytical Solution in The One-Dimensional Ising Model

1. Ising Model

Ising Model은 강자성(ferromagnetic) 모델로 매우 단순한 고체 모형 중 하나이다.

(1) 모든 전자는 up spin(+1)과 down spin(-1), 즉 2가지의 상태만 가진다.

(2) 모든 전자는 가장 주변 원자(nearest-neighbor)와만 상호작용한다.

위 두가지 규칙에 입각하여 만들어진 고체모형으로 간단하지만 기초적인 모형이다. 이 모형은 아니라, 사회과학, 경제학 등 여러 분야에서 응용되어 사용된다.

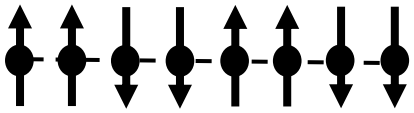


Fig.1.1. One-dimensional Ising model diagram

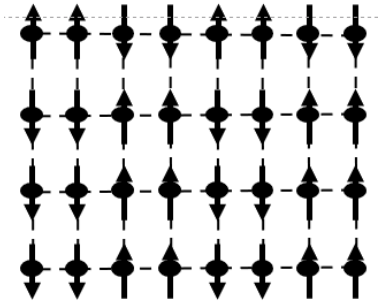


Fig 1.2. Two-dimensional Ising model diagram

이웃한 원자 사이의 상호작용만 고려하는 Ising Model에는 2가지 type의 link가 있다. 스핀의 방향이 같은 link는 Parallel link, 스핀의 방향이 다른 link를 Antiparallel link이라 한다. Fig 1.3.을 살펴보면 왼쪽 2개의 그림이 Parallel link이고, 나머지 오른쪽 2개 그림이 Antiparallel link이다. 이 link는 에너지 계산에 매우 중요하게 사용되며 Parallel links의 에너지는 -J이고, Antiparallel links의 에너지는 +J이다.

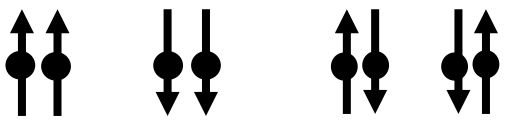


Fig 1.3. Parallel links and Antiparallel links diagram

이를 다수의 전자가 존재하는 상황으로 확장하면 Magnetization 과 Interaction Energy(E_i)를 구할 수 있다. 먼저 Magnetization은 물질이 갖는 magnetic moment로, 스핀이 같은 방향으로 정렬되어 있는 전자들이 많을수록 (무질서도가 작을수록) 큰 magnetization을 갖게 되고, 이는 곧 ferromagnetic을 의미한다.

Ising model에서 magnetization은 up spin의 개수와 down spin의 개수의 차의 절대값이 클수록 magnetization이 크다고 해석할 수 있다. 따라서 그 관계는 Eqn (1.1)과 같다.

$$\boldsymbol{magnetization\ M = \mu\,|(N_+ - N_-)|} \qquad (1.1)$$

내부 에너지는 spin의 방향이 같을 경우 -J, 다를 경우 +가 된다.

$$\boldsymbol{interaction\ energy\ E_i = -J(N_{++} + N_{--} - N_{+-} - N_{-+})} \qquad (1.2)$$

Eqn (1.2)를 보면

N_{++}는 up spin과 up spin이 이웃한 개수를,

N_{--}는 down spin과 down spin이 이웃한 개수를,

N_{+-}는 up spin과 down spin이 이웃한 개수를,

N_{-+}는 down spin과 up spin이 이웃한 개수를 의미한다.

위 식을 통해 interaction Energy를 구할 수 있으며, 이를 이용하여 다른 물리 량을 도출할 수 있다. 이 과정은 후술하고자 한다.

2. Partition Function

One-dimensional Ising model에서 Partition function을 유도할 수 있다.

Partition function을 유도하기 전, Partition function에 필요한 에너지의 정의는 Eqn (1.3)과 같다.

$$E(\{\sigma_i\}) = -J\sum_{i=1}^{N}\sigma_i\sigma_{i+1} - \mu B\sum_{i=1}^{N}\sigma_i \qquad (1.3)$$

Eqn (1.3)의 Parameter들은 다음과 같다.

$\mu: magnetic\ moment$
$J: coupling\ constant$
$B:\ external\ magnetic\ field$
$\beta = 1/k_BT$

Partition Function Z는 Eqn(1.4)로 정의되며

$$Z = \sum_{\{\sigma_i\}} e^{-\beta E(\{\sigma_i\})} = \sum_{\sigma_1}\cdots\sum_{\sigma_N} exp[\beta\sum_{i=1}^{N} J\sigma_i\sigma_{i+1} + \frac{\mu}{2}(\sigma_i + \sigma_{i+1})] \qquad (1.4)$$

여기서 Periodic boundary condition이 적용되었다.

Periodic boundary condition이란, 가장 자리에 있는 전자들의 상호작용을 계산하기 위하여 반대편 가장 자리의 있는 전자들과 상호작용한다고 가정하는 것이다. 예를 들어 왼쪽 가장 자리의 전자의 상호작용을 계산하고 싶을 때, 반대편의 가장 오른쪽 자리의 전자가 왼쪽에 있다고 가정한 후 상호작용을 계산하여 그 값을 도출하는 것을 말한다. 이를 통해, 상호작용을 보다 간단하게 계산할 수 있고, matrix 사이즈를 무한으로 늘릴 수 있다는 장점을 가진다.

Partition function의 값을 쉽게 표현하기 위해서 대각화(diagonalization)를 이용한다.

$$A(\sigma_i, \sigma_{i+1}) = exp[\beta\sum_{i=1}^{N} J\sigma_i\sigma_{i+1} + \frac{\mu}{2}(\sigma_i + \sigma_{i+1})] \qquad (1.5)$$

$$Z = \sum_{\sigma_1}\cdots\sum_{\sigma_N} A_{\sigma_1,\sigma_2}A_{\sigma_2,\sigma_3,}\cdots A_{\sigma_{N-1},\sigma_N}A_{\sigma_N,\sigma_1} = tr(A^N) = tr\left(A^N_{diag}\right) = \begin{pmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{pmatrix}^N = \lambda_1^N + \lambda_2^N \qquad (1.6)$$

라고 하면,

$$|A - \lambda I| = \begin{vmatrix} e^{\beta(J+\mu B)} - \lambda & e^{-\beta J} \\ e^{-\beta J} & e^{\beta(J-\mu B)} - \lambda \end{vmatrix} = 0 \qquad (1.7)$$

$$\lambda = e^{\beta J}\cosh(\beta\mu B) \pm \sqrt{e^{2\beta J}\cosh^2(\beta\mu B) - 2\sinh(2\beta J)} \qquad (1.8)$$

$if\ \lambda_1 > \lambda_2\ then\ \lambda_1^N \gg \lambda_2^N,$

$$\boldsymbol{Z \approx \lambda_1^N = \left[e^{\beta J}cosh(\beta\mu B) + \sqrt{e^{2\beta J}cosh^2(\beta\mu B) - 2\,sinh(2\beta J)}\right]^N} \qquad (1.9)$$
$for\ N \to \infty$

3. Energy, Heat Capacity, Magnetization and Susceptibility

이번에는 Partition function의 결과를 이용하여 Entropy, Heat Capacity, Magnetization의 Exact solution을 구해보았다.

먼저 Helmholtz Free energy를 구해보면,

$$F = -k_BTln[Z] \qquad (1.10)$$

으로 정의되므로

$$\boldsymbol{F = -Nk_BTln[e^{\beta J}\cosh(\beta\mu B) + \sqrt{e^{2\beta J}\cosh^2(\beta\mu B) - 2\sinh(2\beta J)}]} \qquad (1.11)$$

이다.

Entropy at B = 0

B, 즉 외부자기장이 0인 상태에서 Entropy의 Exact solution을 구한다.

B=0인 상태에서 Free energy를 다시 구해보면,

$$F = -Nk_BTln\left[e^{\beta J} + e^{-\beta J}\right] = -Nk_BTln[2cosh\beta J] \quad (1.12)$$

이고

$$\boldsymbol{entropy\ S = -\frac{\partial F}{\partial T}} = \boldsymbol{Nk_B\left[ln\left(2cosh\frac{J}{k_BT}\right) - \frac{J}{k_BT}\left(tanh\frac{J}{k_BT}\right)\right]} \quad (1.13)$$

-1. Heat Capacity

Heat Capacity (Cv)는 위의 Entropy와 T에 대해 partial differential의 관계를 가진다. 따라서 Heat Capacity를 구해보면 Eqn (1.14)와 같다.

$$\boldsymbol{C_V = T\frac{\partial S}{\partial T} = \frac{NJ^2}{k_BT^2}sech^2\frac{J}{k_BT}} \quad (1.14)$$

-2. Magnetization

Magnetization은 위의 Free energy와 B에 대한 partial differential의 관계를 가진다.

$$M = -\frac{\partial F}{\partial B} = \beta\mu Nk_BT\frac{e^{\beta J}\sinh\beta\mu B + \frac{e^{2\beta J}\sinh\beta\mu B\cosh\beta\mu B}{\sqrt{e^{2\beta J}\cosh^2\beta\mu B - 2\sinh 2\beta J}}}{e^{\beta J}\cosh\beta\mu B + \sqrt{e^{2\beta J}\cosh^2\beta\mu B - 2\sinh 2\beta J}} \quad (1.15)$$

약한 magnetic field에서 $\boldsymbol{sinh\beta\mu B \approx 1, cos\beta\mu B \approx 1}$ 로 근사 할 수 있으므로,

$$\boldsymbol{M \approx N\mu^2\beta\, e^{2\beta J}B = \frac{N\mu^2}{k_BT}\, e^{\frac{2J}{k_BT}}B} \quad (1.16)$$

-3. Susceptibility

$$\boldsymbol{\chi = \mu_0\frac{N\mu^2}{k_BT}\, e^{\frac{2J}{k_BT}}B} \quad (1.17)$$

-4. Graph

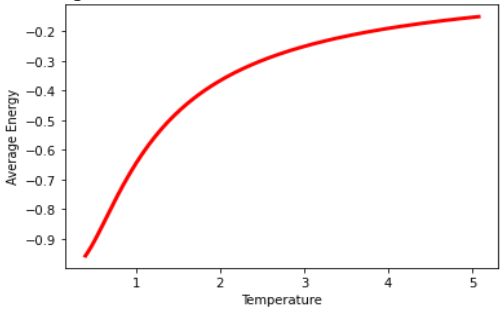


Fig 1.4. Analytical average energy graph in the one-dimensional Ising model

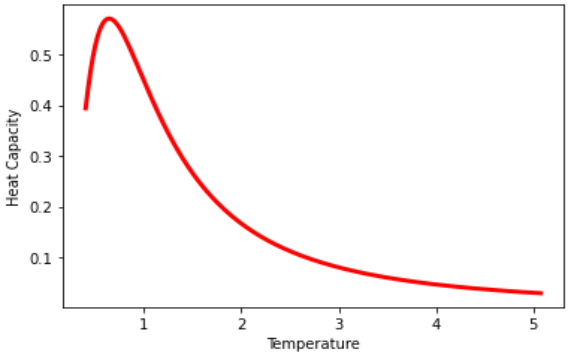


Fig 1.5. Analytical heat capacity graph in the one-dimensional Ising model

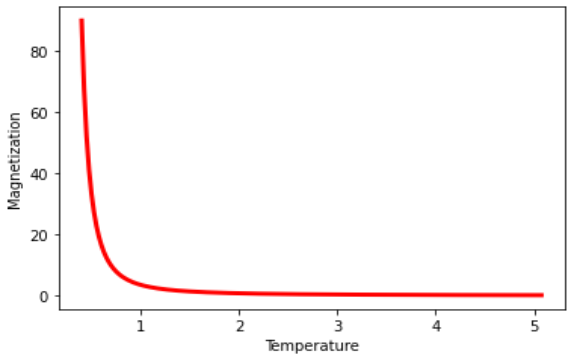


Fig 1.6. Analytical magnetization graph in the one-dimensional Ising model

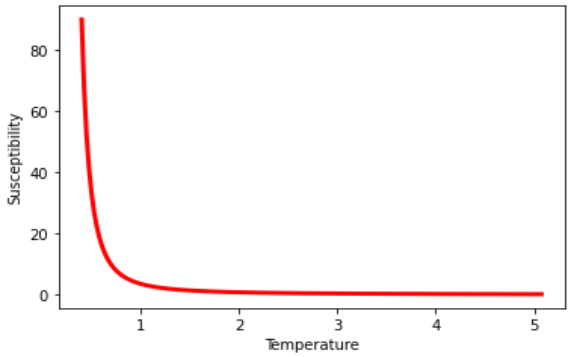


Fig 1.7. Analytical susceptibility graph in the one-dimensional Ising model

Analytical Solution의 결과는 Fig 1.4.을 통해 Average Energy를, Fig 1.5.을 통해 Heat Capacity를, Fig 1.6.을 통해 Magnetization을, Fig 1.7.을 통해 Susceptibility를 알 수 있다.

그래프 개형을 보면, Temperature (T)가 증가할수록 Average Energy가 증가하는 경향을 보인다. Heat Capacity는 T가 증가할 때, 계속 증가하는 것이 아닌, 1 부근에의 point전까지만 증가하고 이후에는 감소하는 global maximum (T의 온도 범위가 0에서 5로 한정되어 있어 있지만, 식을 분석해볼 경우 point 이후에서는 반드시 Heat Capacity가 감소하여야 한다.)을 가진다.

Ferromagnetism을 확인하기 위한 Ising model에서 가장 중요한 Magnetization의 그래프의 개형을 보면 Critical Point를 가지지 않고, 감소하는 경향을 보인다.

이를 통해 One-dimensional Ising model은 상전이(Phase transition)를 설명할 수 없고, 실제 모델을 분석하는 것에 사용할 수 없음을 알 수 있다.

II. Numerical Solution in The One-Dimensional Ising Model

Ⅱ 에서는 Monte Carlo Method와 Markov Chain에 대해 설명하고, 이를 응용한 Markov Chain Monte Carlo와 MCMC의 하나의 방법인 Metropolis Algorithm을 설명한다. 이를 이용하여 One-Dimensional Ising model의 Numerical solution을 구한 후, 그 구한 결과를 앞의 Analytical Solution의 Graph와 비교하여 결과의 정확성을 확인하고자 한다.

1. Monte Carlo Method

Monte Carlo Method는 랜덤 샘플링의 반복을 통하여 approximation을 얻어 내는 방법을 말한다.

이 Monte Carlo Method를 구현한 간단한 예시 중, π approximation이 있다.
$x = [x|-1 \le x \le 1], y = [y| -1 \le y \le 1]$의 정사각형이 있을 때, 공을 정사각형 내부에 Randomly하게 던진다고 생각한다. 반지름이 1이고 중심이 origin에 있는 원이 있을 때, 원 안에 공이 들어가면 counting 한다. 이를 계속 반복한다.

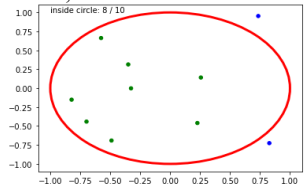
$$\frac{the\ number\ of\ ball\ in\ Circle}{the\ number\ of\ ball\ in\ Square} = \frac{Area\ of\ Circle}{Area\ of\ Square} = \frac{\pi \times 1^2}{2 \times 2} \quad (2.1)$$

$$Thus, \pi = 4 \times \frac{the\ number\ of\ ball\ in\ Circle}{the\ number\ of\ ball\ in\ Square} \quad (2.2)$$

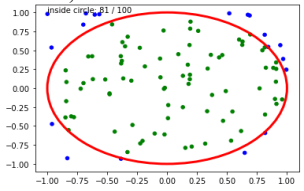
이므로 π을 샘플링의 반복을 통하여 구할 수 있다.
Monte Carlo Method에서, Monte Carlo 샘플링의 횟수가 증가할수록, 특정한 값에 수렴하며 점점 exact value에 근접한다는 특징이 있다.

N을 Monte Carlo 샘플링 횟수라고 한다면,

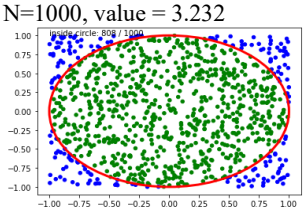
N=10 , value = 3.2



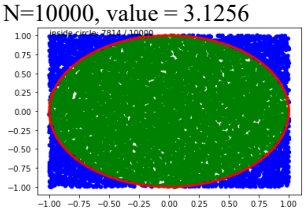
N=100, value = 3.24



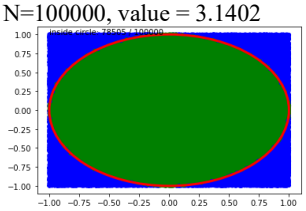
N=1000, value = 3.232



N=10000, value = 3.1256



N=100000, value = 3.1402



N=1000000, value = 3.141304

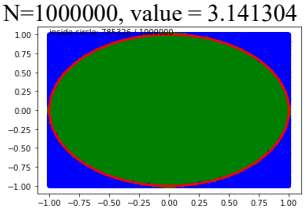


Fig 2.1. pi value calculated by Monte carlo method

N	10	100	1000	10000	100000	1000000
Pi value	3.2	3.24	3.232	3.1256	3.1402	3.141304
Relative Error(%)	1.9	3.13	2.878	0.5090	0.0443	0.009188

Fig 2.2. Pi value and relative error according to Monte Carlo sampling number

의 결과로 Monte Carlo Sampling의 수가 증가할수록 Relative Error가 줄어 참 값에 가까워진다.

Python Code는 다음과 같다.

```python
import random, math
import numpy as np
import matplotlib.pyplot as plt


def pi_value(number):

    in_circle_number = 0
    data_x = []
    data_y = []

    # Repeat throwing particles
    for i in range(number):
        x = random.uniform(-radius, radius)
        y = random.uniform(-radius, radius)


        distance = math.sqrt(x**2 + y**2)
        #If the particle is inside circle, add 1 to in_circle_number.
        if distance < radius:

            in_circle_number += 1

        data_x.append(x)
        data_y.append(y)

    return in_circle_number, data_x, data_y

number = 10000
radius = 1

pi_temp, result_x, result_y = pi_value(number)


pi = 4 * pi_temp / (number)

print('pi value: ', pi)
```

Fig 2.3.(a) Python code calculating pi by Monte Carlo – part1 (calculating value)

```
color = []

# If the particle is in circle, color of particle is green.
# Otherwiese, if the particle is out of circle, color of particle is blue.

for i in range(number):
    distance = math.sqrt(result_x[i]**2 + result_y[i]**2)
    if distance < radius:
        color.append('g')
    else:
        color.append('b')

circle = plt.Circle((0,0), radius, color='r', fill=False, linewidth=3)
flg, ax = plt.subplots()
ax.add_patch(circle)

plt.scatter(result_x, result_y, c=color, s=20)

plt.text(-radius, radius, ('inside circle: '+ str(pi_temp)+' / '+str(number)))

plt.show()
```

Fig 2.3.(b) Python code calculating pi by Monte Carlo - part2 (graph)

1. Markov Chain (MC)

Markov Chain Model은 확률모델로, 오직 다음 상황을 결정함에 있어 바로 직전의 상황만 고려하는 모델이다.

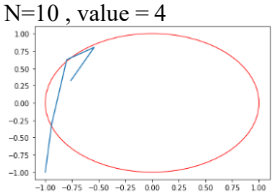
Markov Chain으로 간단히 구현 가능한 예시 또한 π approximation이 있다.

$x = [x|-1 \le x \le 1], y = [y| -1 \le y \le 1]$ 범위의 정사각형이 있을 때, 좌표 (-1, -1)에서 시작한다. -1에서 1사이의 랜덤 순자를 뽑아서 각각의 x와 y 좌표에 더한다. 만약 뽑은 랜덤 숫자를 더한 새로운 좌표가 정사각형 안에 존재한다면 그 좌표는 다음 좌표가 된다. 만약 정사각형 안에 존재하지 않는다면, 원래 좌표가 다음 좌표가 된다.

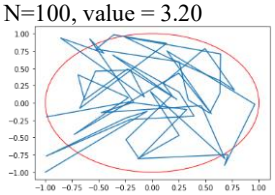
이렇게 좌표를 기록한 다음, 좌표별로 계산하여 반지름이 1이고 중심이 origin에 있는 원 안에 좌표가 들어오면 counting한다.

$$\pi = 4 \times \frac{counted\ number}{total\ trial} \qquad (2.3)$$

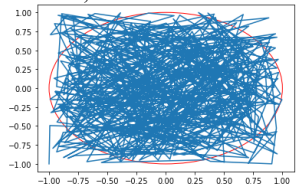
N=10 , value = 4



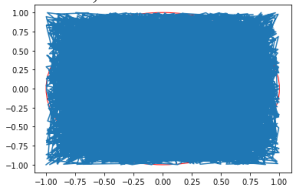
N=100, value = 3.20



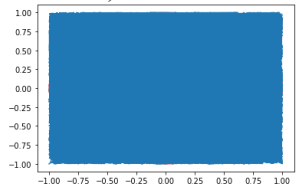
N=1000, value = 2.952



N=10000, value = 3.15



N=100000, value = 3.15632



N=1000000, value = 3.145240

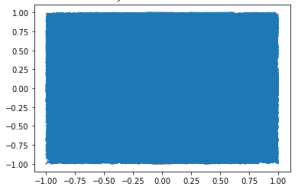


Fig 2.4. pi value calculated by Markov Chain

N	10	100	1000	10000	100000	1000000
Pi value	4.0	3.2	2.952	3.150	3.1563	3.14524
Relative Error(%)	27	1.85	6.034	0.268	0.46878	0.11609

Fig 2.5. Pi value and relative error according to Markov Chain sampling number

Markov Chain 또한 횟수가 증가하면 참 값에 근접함을 알 수 있다.

Python code는 다음과 같다.

```
import math, random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def throwing(number):
    #Initial Coordinate
    x = -1
    y = -1

    coordinates = []
    for i in range(number):
        x_add = random.uniform(-1, 1)
        y_add = random.uniform(-1, 1)

        if abs(x + x_add) < 1 and abs(y + y_add) < 1:
            x = x + x_add
            y = y + y_add
        else:
            pass

        coordinates.append([x,y])
    return coordinates

def counting(coordinates):
    count_number = 0

    for i in coordinates:
        x, y = i
        distance = math.sqrt(x**2 + y**2)
        if distance < 1:
            count_number += 1

    return count_number

result = []
number = 10000 # number of tryiing

for i in range(5):
    result_matrix = throwing(number)
    number_hit = counting(result_matrix)
    pi = 4 * number_hit / number
    result.append(pi)

index = ["1", "2", "3", "4", "5"]
column = ["pi"]

pd.DataFrame(result, index, column)
```

Fig 2.6.(a) Python code calculating pi by Markov Chain – part2 (calculating value)

```
# graph
result_matrix = throwing(number)
x_graph = []
y_graph=[]

for i in result_matrix:
    x, y = i
    x_graph.append(x)
    y_graph.append(y)

circle = plt.Circle((0,0), 1, color='r', fill=False, linewidth=1)
flg, ax = plt.subplots()
ax.add_patch(circle)

plt.plot(x_graph,y_graph)
```

Fig 2.6.(b) Python code calculating pi by Markov Chain -part2 (graph)

2. Markov Chain Monte Carlo (MCMC) and Metropolis Algorithm

-1. Markov Chain Monte Carlo (MCMC)

앞의 1. Monte Carlo와 2. Markov Chain을 융합시킨 Markov Chain Monte Carlo에 대해 설명하고자 한다. 명칭이 길기 때문에 이후에는 각 단어의 앞 글자를 가져온 MCMC로 부르겠다. MCMC는 Markov Chain과 Monte Carlo의 특성을 두가지 가지고 있는 알고리즘으로, 반복된 랜덤 샘플링(Monte Carlo)과, 이전 상황만으로 다음 상황을 결정(Markov Chain)하는 알고리즘이다.

MCMC의 메커니즘을 이용하면서 간단하지만 주요하게 사용되는 Algorithm중 Metropolis Algorithm이 있다.

-2. Metropolis Algorithm

Metropolis Algorithm의 과정은 다음과 같다.

먼저 초기값을 뽑는다 (Select initial value)

$$x_0 \sim \pi(x) \tag{2.4}$$

밑의 과정을 반복한다 (Repeat, i=1,2,3…)

- 테스트 Sample을 제작한다 (Generate test sample)

$$x^* \sim q(x_{i+1}|x_i) \tag{2.5}$$

- α값을 정의한다.

$$\alpha(x^*, x_i) = \min\ \{1,\ \frac{\pi(x^*)}{\pi(x_i)}\} \tag{2.6}$$

- 0에서 1사이의 난수를 생성한다 (Generate uniform random number)

$$\mathrm{u} \sim U(0,\ 1) \tag{2.7}$$

- u와 α값을 비교하여 다음 값을 정한다.

$$if\ u < \alpha \qquad x_{i+1} = x^* \tag{2.8a}$$
$$else \qquad x_{i+1} = x_i \tag{2.8b}$$

-3. Application Metropolis Algorithm to The One-Dimensional Ising Model

Metropolis Algorithm을 The One-Dimensional Ising Model에 적용하는 과정을 설명하겠다.

개략적으로 Metropolis Algorithm에서의 초기값을 무작위 random coordinate에서 Boltzmann distribution을 이용하여 $\pi(a)$로 설정하고, Flip을 시켜 $\pi(b)$를 구하고, 이를 Metropolis Algorithm의 과정대로 비교하여 반복하게 된다.

먼저 초기값을 뽑는다 (Select initial value)

$$x_0 \sim \pi(a) \tag{2.9}$$

초기값은 Ising model의 좌표 중에서 랜덤으로 골라 설정한다. x_0는 $\pi(a)$가 되고, $\pi(a)$는 Boltzmann distribution에 따라서 Eqn (2.12)와 같이 설정한다.

$$\pi(a) = \frac{e^{\beta E_a}}{Z} \tag{2.10}$$

밑의 과정을 반복한다 (Repeat, i=1,2,3…)

- Randomly하게 선택한 좌표와 상호작용하는 주변 전자들의 spin방향을 Flip 시킨다.

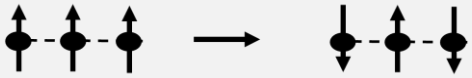


Fig 2.7. Flip in Metropolis algorithm

따라서 Flip된 후의 상황을 $\pi(b)$라고 하면,

$$\pi(b) = \frac{e^{\beta E_b}}{Z} \tag{2.11}$$

- α값을 정의한다.

$$\alpha(x^*, x_i) = min\ \left\{1, \frac{\pi(b)}{\pi(a)}\right\} = \min\{1, e^{\beta(E_b - E_a)}\} \tag{2.12}$$

- 0에서 1사이의 난수를 생성한다 (Generate uniform random number)

$$\mathrm{u} \sim U(0,\ 1) \tag{2.13}$$

- u와 α값을 비교하여 다음 값을 정한다.

$$if\ u < \alpha \qquad x_{i+1} = x^* \tag{2.14a}$$
$$else \qquad x_{i+1} = x_i \tag{2.15b}$$

이는 즉, Flip 여부를 결정하게 된다.

3. Detailed Balance

-1. Detailed Balance

Detailed Balance란 MCMC와 Metropolis Algorithm의 성립에 있어서 가장 중요한 요소이다.

만약 Randomly하게 Sample을 만들고, 이를 조건에 대해 값을 구하려고 하는데 어떤 조건에 대해 특정한 값으로 수렴하지 않는다면 이 Algorithm은 의미가 없다.

따라서 어떤 특정 조건이 주어졌을 때, 그 값이 수렴하기 위해서는 Detailed Balance조건 (Global

Balance 조건)을 만족해야 한다.
간단한 Diagram으로 설명하겠다.

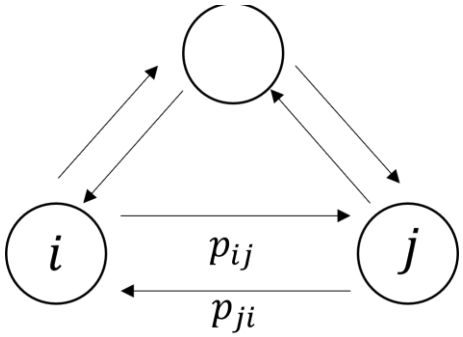


Fig 2.7. Diagram explaining detailed balance

Detailed balance equation은 Eqn과 같다.

$$\boldsymbol{\pi_i p_{ij} = \pi_j p_{ji}} \ \forall \boldsymbol{i,j} \tag{2.16}$$

왼쪽 항이 의미하는 것은 i에서 j로 흐르는 확률의 양을 말한다. 반대로 오른쪽 항이 의미하는 것을 j에서 i로 흐르는 확률의 양을 말한다.

둘이 같다는 것은 왼쪽에서 오른쪽으로 흐르는 확률의 양이 오른쪽에서 왼쪽으로 흐르는 확률의 양과 같으며, 즉 i, j만 포함된 system에서는 확률의 양이 흐르지 않는, 즉 확률의 양이 변함이 없다는 것과 같다. 이는 고정된 값이 도출됨을 보장하며, Repeating할 때 계속해서 달라지는 것이 아닌, 일정한 값으로의 수렴을 보장할 수 있다.

System의 모든 지점 들에서 i와 j를 설정했을 때, 이 관계를 모두 만족하면 그때 Detailed balance를 만족하며 Stationary Distribution하다고 할 수 있다. 이는 MCMC 혹은 Metropolis algorithm을 하였을 때, 조건에 맞는 값들이 수렴하여 원하는 Numerical solution을 얻을 수 있음을 보장한다.

-2. Proof that Metropolis Algorithm calculating one-dimensional Ising model satisfies Detailed balance

먼저

$$\boldsymbol{case\ 1)\ \pi(a) > \pi(b)} \tag{2.17}$$

$$\boldsymbol{p(a \to b) = min\left[1, \frac{\pi(b)}{\pi(a)}\right] = \frac{\pi(b)}{\pi(a)}} \tag{2.18}$$

$$\boldsymbol{\Rightarrow \pi(a)p(a \to b) = \pi(b)} \tag{2.19}$$

$$\boldsymbol{p(b \to a) = min\left[1, \frac{\pi(a)}{\pi(b)}\right] = 1} \tag{2.20}$$

$$\boldsymbol{\Rightarrow \pi(b)p(b \to a) = \pi(b)} \tag{2.21}$$

$$\boldsymbol{\therefore \pi(a)p(a \to b) = \pi(b)P(b \to a)} \tag{2.22}$$

$$\boldsymbol{case\ 2)\ \pi(a) < \pi(b)} \tag{2.23}$$

$$\boldsymbol{p(a \to b) = min\left[1, \frac{\pi(b)}{\pi(a)}\right] = 1} \tag{2.24}$$

$$\boldsymbol{\Rightarrow \pi(a)p(a \to b) = \pi(a)} \tag{2.25}$$

$$\boldsymbol{p(b \to a) = min\left[1, \frac{\pi(a)}{\pi(b)}\right] = \frac{\pi(a)}{\pi(b)}} \tag{2.26}$$

$$\boldsymbol{\Rightarrow \pi(b)p(b \to a) = \pi(a)} \tag{2.27}$$

$$\boldsymbol{\therefore \pi(a)p(a \to b) = \pi(b)P(b \to a)} \tag{2.28}$$

4. Energy, Heat Capacity, Magnetization and Susceptibility

여기서 E_i는 Hamiltonian과 같은 물리량이고, Hamiltonian은 H로 줄여 작성하였다.
N은 matrix size이고, MC는 Monte Carlo 시행 횟수이다.

$$hamiltonian\ H = -J(N_{++} + N_{--} - N_{+-} - N_{-+}) \tag{2.29}$$

-1. Average Energy

$$< E > \ = \frac{1}{N}\frac{1}{MC} < H > \tag{2.30}$$

에너지 식은 다음과 같다. 여기서는 repeat을 하여 Hamiltonian값을 누적하기 때문에, repeat 횟수와 Monte Carlo 시행횟수 요인을 제거해주기 위해서 N과 MC로 나누고, Average Energy를 구하게 된다.

-2. Heat Capacity

$$\begin{aligned} < C_V > &= \frac{\partial < E >}{\partial T} = \frac{\partial \beta}{\partial T}\frac{\partial < E >}{\partial \beta} \\ &= -\beta^2 \frac{\partial < E >}{\partial \beta} = \beta^2 \frac{\partial^2}{\partial \beta^2} logZ \\ &= \beta^2(< E^2 > \ - \ < E >^2) \end{aligned} \tag{2.31}$$

-3. Magnetization

$$< M >= \frac{1}{N}\frac{1}{MC} | \ (N_+ - \ N_-)| \tag{2.32}$$

Average Energy와 같은 이유로 N과 MC로 나눈다.

-4. Susceptibility

$$\begin{aligned} < S > &= \frac{\partial < M >}{\partial T} = \frac{\partial \beta}{\partial T}\frac{\partial < M >}{\partial \beta} = -\beta^2 \frac{\partial < M >}{\partial \beta} \\ &= \beta^2(< M^2 > \ - \ < M >^2) \end{aligned} \tag{2.33}$$

5. Python Code

Part1.

```python
import random, math
import matplotlib.pyplot as plt
import numpy as np

global n, J, N, kb

n = 20
J = 1
kb = 1.3
```

Fig 2.7.(a) One-dimensional Ising model Python code- part1 (import and define basic value)

이 코드에서는 random, math, matplotlib, numpy 함수를 불러온다. n은 matrix size이고, J=1로 고정한다.

Part2.

```python
def mk_matrix_1d():
    data_init = np.zeros([n])
    data_init = np.random.choice([-1,1], size = (n))
    return data_init
```

Fig 2.7.(b) One-dimensional Ising model Python code- part2 (make random one-dimensional matrix)

-1과 1로 무작위의 random matrix를 만든다.

Part3.

```python
def pi_value_1d(data_unit, beta):

    Energy = 0

    x = random.randint(0,(n-1))

    right = data_unit[(x+1)%n]
    left = data_unit[(x-1)%n]
    Energy += data_unit[x] * (right + left)

    pi_a = math.exp((+1)*beta*Energy)
    pi_b = math.exp((-1)*beta*Energy)
    pi_divided = pi_b / pi_a

    if pi_divided <= 1:
        if random.random() < pi_divided:
            data_unit[x] = -data_unit[x]
    elif pi_divided > 1:
        data_unit[x] = -data_unit[x]

    return data_unit
```

Fig 2.7.(c) One-dimensional Ising model Python code- part3 (determine whether to flip)

위의 Metropolis Algorithm에 따라서 Flip여부를 결정한다.

Part4.

```python
def matrix_repeat_1d(number, matrix_matrix):
    for i in range(number):
        matrix_matrix = pi_value_1d(matrix_matrix, 1/0.1)
    return matrix_matrix
```

Fig 2.7.(d) One-dimensional Ising model Python code- part4 (repeat Monte Carlo to make random matrix)

Monte Carlo 방법으로 matrix를 더 randomly하게 만들어준다. 이미 random으로 만든 함수를 더 randomly하게 섞어 주는 것은 컴퓨터 함수가 random함수임에도 불구하고 규칙이 있는 경우가 종종 발생하기 때문이다. 이를 막기 위하여 Flip한다.

Part5.

```python
def cal_temperature(matrix_cal, beta):

    Sum_Energy = 0
    Sum_Energy_square = 0
    Avg_Energy = 0
    Avg_Energy_square = 0
    Sum_M_square = 0
    Sum_M = 0
    Avg_M=0
    Avg_M_square = 0

    Sum_Hc = 0
    Sum_Sus = 0

    #Repeat N times

    for j in range(N):

        H = 0
        H_square = 0

        #Calculate Hamiltonian

        for x in range(n):
            right = matrix_cal[(x+1)%n]
            left = matrix_cal[(x-1)%n]

            h_temp =  -J * (matrix_cal[x] * (right + left)) / 2

            H += h_temp / (n)
            H_square += h_temp**2 / (n)

        Sum_Hc += beta * beta * (H_square - H*H)

        M = np.sum(matrix_cal)

        Sum_Energy += H
        Sum_Energy_square += H_square
        Sum_M += abs(M)
        Sum_M_square += M*M
        matrix_cal = pi_value_1d(matrix_cal, beta)

    #Get average values by dividing N

    Avg_Energy = Sum_Energy / N
    Avg_Energy_square = Sum_Energy_square / N
    Avg_M = beta * Sum_M / N
    Avg_M_square = beta * Sum_M_square / N
    Avg_Hc = Sum_Hc / N * 3
    Sus = - (beta**2) * ((Avg_M_square) - Avg_M*Avg_M) /(n)

    return Avg_Energy, Avg_M, Avg_Hc, Sus, matrix_cal
```

Fig 2.7.(e) One-dimensional Ising model Python code- part5 (calculate Energy, Heat Capacity, Magnetization, Susceptibility in certain temperature)

특정 온도(T)에서 Average Energy, Heat Capacity, Magnetization, Susceptibility를 계산한다.

Part6.

```python
result_Avg_Energy = []
result_Hc = []
result_Sus = []
result_Mg = []
result_matrix = []

for i in range(len(T)):
    temp1, temp2, temp3, temp4, temp5 = cal_temperature(matrix_cel, 1/(kb*T[i]))
    result_Avg_Energy.append(temp1)
    result_Mg.append(temp2)
    result_Hc.append(temp3)
    result_Sus.append(temp4)
```

Fig 2.7.(f) One-dimensional Ising model Python code- part6 (Calculate properties from 0.4 to 5)

Part5에서 define된 함수를 이용하여 0.4에서 5까지 T에 따른 Average Energy, Heat Capacity, Magnetization, Susceptibility를 계산하여 list에 저장한다.

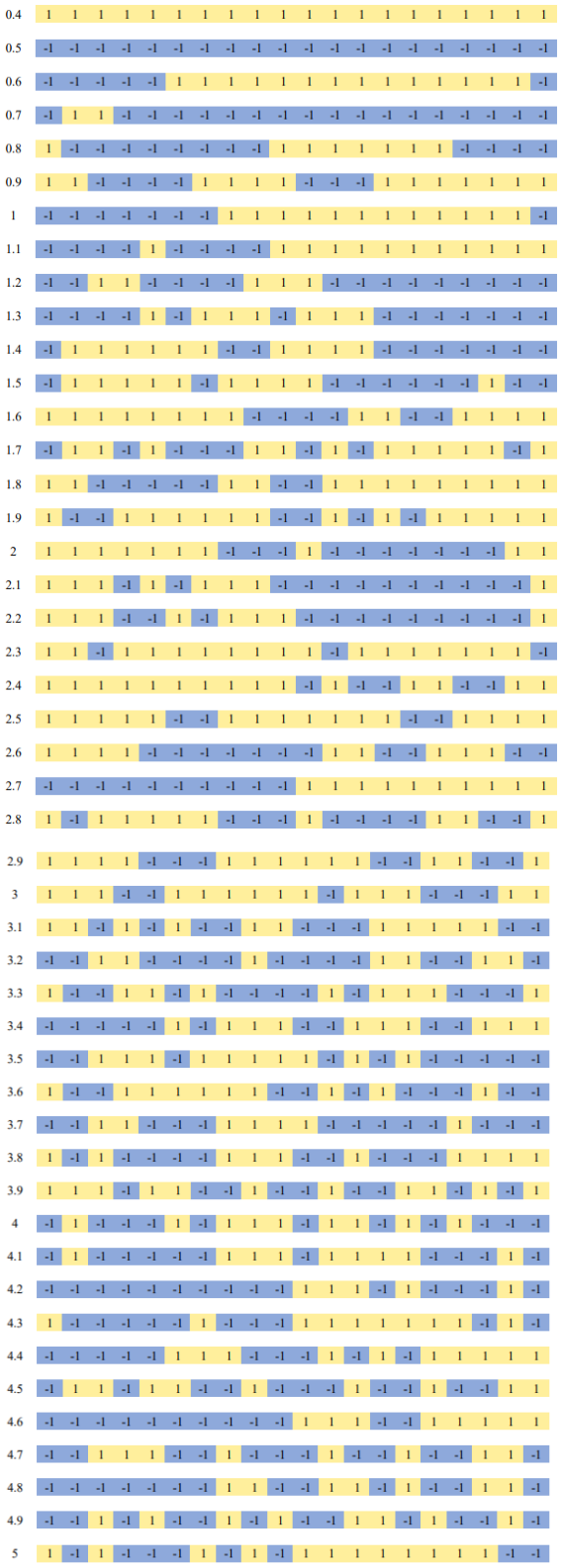
Part7.

```python
j = 0
y_label = ['Average Energy', 'Susceptibility', 'Heat Capacity', 'Magnetization']
for i in [result_Avg_Energy, result_Sus, result_Hc, result_Mg]:
    plt.scatter(T, i)
    plt.xlabel('Temperature')
    plt.ylabel(y_label[j])
    plt.show()
    j += 1
```

Fig 2.7.(g) One-dimensional Ising model Python code- part7 (Graph)

6. Numerical Solution

-1. Figure – distribution of spin according to temperature.



온도에 따른 스핀의 분포 heatmap이다. 1은 노란색, -1은 파란색이며 온도가 낮을수록 주변과 같은 spin을 가지고 있는 걸을 확인할 수 있고, 반대로 온도가 높을수록 주변과 다른 스핀을 가짐으로써 무작위성이 증가함을 볼 수 있다.

(이 Figure에서 보아야 할 것은 노란색과 파란색이 얼마나 있느냐가 아닌, 노란색 주변에 파란색의 존재 여부이다. 관심을 두어야 할 것은 +1의 spin과 -1 spin의 절대적인 양이 아닌, +1 spin 주변에 -1 spin이 없는 지의 상대적인 관계를 보아야 한다.)

-2. Graph

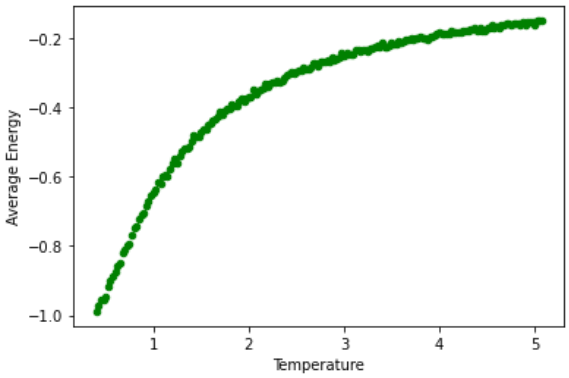


Fig 2.8. Numerical Average Energy graph in the one-dimensional Ising model

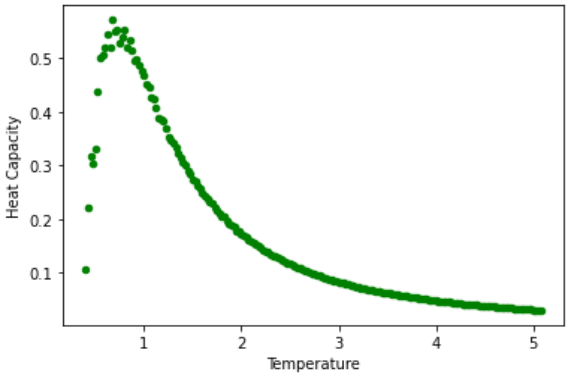


Fig 2.9. Numerical Heat Capacity graph in the one-dimensional Ising model

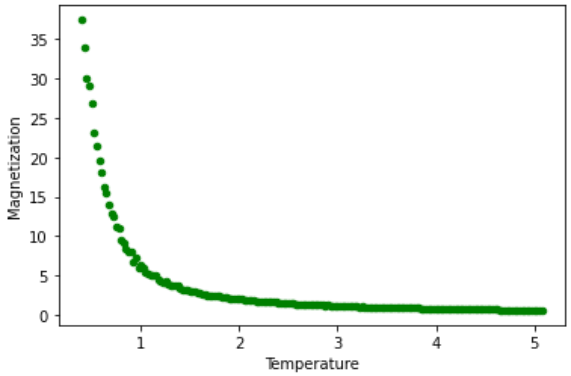


Fig 2.10. Numerical Magnetization graph in the one-dimensional Ising model

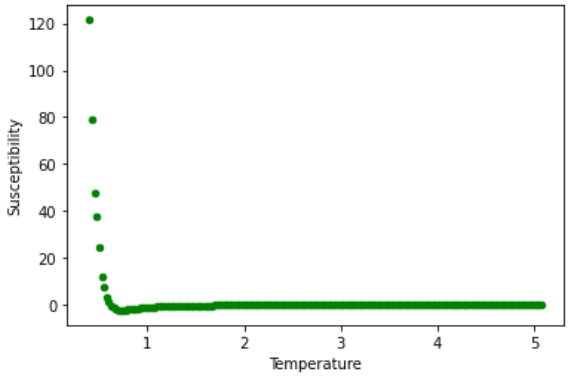


Fig 2.11. Numerical Susceptibility graph in one-dimensional Ising model

결과는 위의 Fig와 같다.

-3. Comparing Numerical Solution with Analytical Solution

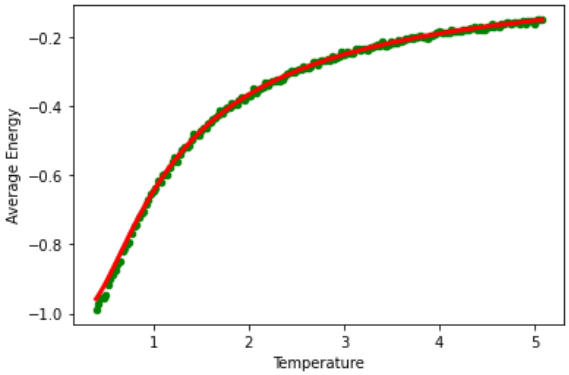


Fig 2.12. Comparing Numerical Solution with Analytical Solution: Average Energy

그래프가 완전하게 일치함을 볼 수 있다.

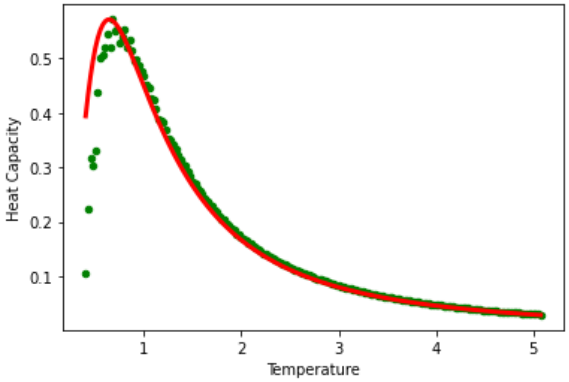


Fig 2.13. Comparing Numerical Solution with Analytical Solution: Heat Capacity

높은 온도로 갈수록 그래프가 일치하나, peak point가 생기는 Temperature가 Numerical Solution이 더 큰 결과로 오차가 발생하였음을 볼 수 있다.

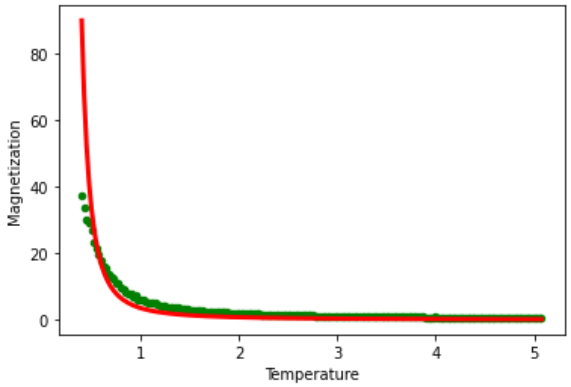


Fig 2.14. Comparing Numerical Solution with Analytical Solution: Magnetization

높은 온도에서는 일치함을 보이나, 약 0.5를 전후로 값이 교차되며 오차가 발생함을 알 수 있다. 특히, Numerical Solution은 약 40에서 최대값이 나오는데 반하여 Analytical Solution은 0에 가까울수록 40을 상회하는 매우 큰 값을 결과로 가짐을 볼 수 있다.

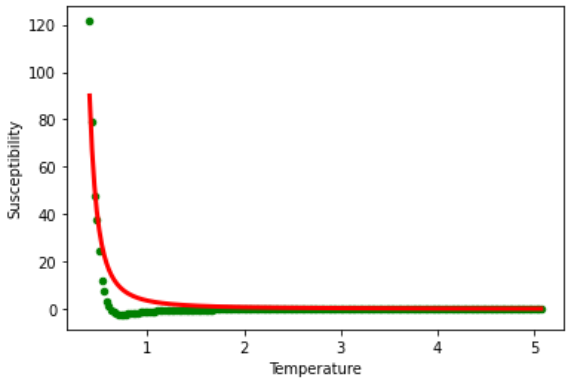


Fig 2.15. Comparing Numerical Solution with Analytical Solution: Susceptibility
약 0.7 전 후로 오차가 발생하였다.

오차분석:
오차를 분석해보면, 먼저 기본적인 물리량 설정에서 오차가 발생하였을 것이다. J=1, kb=1.3으로 둔 것과, 여러가지 변수를 모두 1로 두고 값을 계산하였다. 그래프의 개형을 위주로 보기 위해서 여러 상수 값을 모두 1로 두고 계산하여, 특정 부분에서 오차가 발생하였을 것으로 생각된다.

또한 Numerical Solution은 Approximation으로 참 값에 가깝게 얻는 것이지, 정확한 값을 구하는 것이 아니다. 따라서 오차는 필연적이며, 이 부분에서 생긴 오차가 발생하였을 것으로 생각된다.

III. Numerical Solution in The Two-Dimensional Ising Model

이번에는 Two-dimensional Ising Model에 대하여 분석해보고자 한다. Two-dimensional Ising Model은 Analytical Solution 대신 Numerical Solution만 구하였다. Onsager가 증명한 two-dimensional Ising model의 Analytical Solution을 구하는 과정을 공부해보려 하였으나, 난이도가 상당히 높아, Numerical Solution만 구하고자 한다.

앞의 과정에서 구했던 One-dimensional Ising model을 응용하여 Numerical Solution을 구하고, 이를 그래프를 그려 critical temperature가 도출됨을 확인한다.

One-dimensional Ising model 에서 설명한 부분은 매우 간략하게 넘어가고자 한다.

1. Difference between Numerical Solution in The One-Dimensional Ising Model and Two-Dimensional Ising Model

Two-dimensional Ising model에서 One-dimensional Ising model의 해를 구하는 과정에서 사용하는 MCMC, Metropolis Algorithm의 동일한 과정을 사용한다.

수정해주어야 할 부분은 matrix를 two-dimension으로 바꿔주고, 에너지 계산 시 One-dimension에서 고려했던 좌, 우 요인에, 위 아래까지 고려하게 해주면 된다.

2. Python Code

Part1.

```python
import random, math
import matplotlib.pyplot as plt
import numpy as np

global n, J, N

n = 10
J = 1
```

Fig 3.1. Two-dimensional Ising model Python code - part1 (import and define basic value)

Part2.

```python
def mk_matrix_2d():
    data_init = np.zeros([n,n])
    data_init = np.random.choice([-1,1], size = (n,n))
    return data_init
```

Fig 3.2. Two-dimensional Ising model Python code – part2 (make random two-dimensional matrix)

Part3.

```python
def pi_value_2d(data_unit, beta):

    Energy = 0

    x = random.randint(0,(n-1))
    y = random.randint(0,(n-1))

    top = data_unit[(y-1)%n, x]
    bottom = data_unit[(y+1)%n, x]
    right = data_unit[y, (x+1)%n]
    left = data_unit[y, (x-1)%n]
    Energy += data_unit[y, x] * (top + bottom + right + left)

    pi_a = math.exp((+1)*beta*Energy)
    pi_b = math.exp((-1)*beta*Energy)
    pi_divided = pi_b / pi_a

    if pi_divided <= 1:
        if random.random() < pi_divided:
            data_unit[y,x] = -data_unit[y,x]
    elif pi_divided > 1:
        data_unit[y,x] = -data_unit[y,x]

    return data_unit
```

Fig 3.3 Two-dimensional Ising model Python code – part3 (determine whether to flip)

Part4.

```python
def matrix_repeat_2d(number, matrix_matrix):
    for i in range(number):
        matrix_matrix = pi_value_2d(matrix_matrix, 1/0.1)
    return matrix_matrix
```

Fig 3.4. Two-dimensional Ising model Python cod – part4 (repeat Monte Carlo to make random matrix)

Part5.

```python
def cal_temperature(matrix_cal, beta):

    Sum_Energy = 0
    Sum_Energy_square = 0
    Avg_Energy = 0
    Avg_Energy_square = 0

    Sum_M_square = 0
    Sum_M = 0
    Avg_M=0
    Avg_M_square = 0

    Sum_Hc = 0
    Sum_Sus = 0

    #Repeat N times

    for j in range(N):

        H = 0
        H_square = 0

        #Calculate Hamiltonian

        for x in range(n):
            for y in range(n):
                top = matrix_cal[(y-1)%n, x]
                bottom = matrix_cal[(y+1)%n, x]
                right = matrix_cal[y, (x+1)%n]
                left = matrix_cal[y, (x-1)%n]

                h_temp =  -J * (matrix_cal[y, x] * (top + bottom + right + left)) / 2

                H += h_temp / (n*n)
                H_square += h_temp**2 / (n*n)

        Sum_Hc += beta * beta * (H_square - H*H)

        M = np.sum(matrix_cal)

        Sum_Energy += H
        Sum_Energy_square += H_square
        Sum_M += abs(M)
        Sum_M_square += M*M
        matrix_cal = pi_value_2d(matrix_cal, beta)

    #Get average values by dividing N

    Avg_Energy = Sum_Energy / N
    Avg_Energy_square = Sum_Energy_square / N

    Avg_M = Sum_M / N
    Avg_M_square = Sum_M_square / N

    Avg_Hc = Sum_Hc / N

    Sus = beta * ((Avg_M_square) - Avg_M*Avg_M) /(n*n)

    return Avg_Energy, Avg_M, Avg_Hc, Sus, matrix_cal
```

Fig 3.5. Two-dimensional Ising model Python code – part5 (calculate Energy, Heat Capacity, Magnetization, Susceptibility in certain temperature)

```
init = mk_matrix_2d()
matrix_cel = matrix_repeat_2d(5000, init)

N = 5000

#T = np.arage(start temperature, end temperature, interval)
T = np.arange(0.99, 5.01, step=0.01)
```

```
result_Avg_Energy = []
result_Hc = []
result_Sus = []
result_Mg =[]
result_matrix = []

for i in range(len(T)):
    temp1, temp2, temp3, temp4, temp5 = cal_temperature(matrix_cel, 1/T[i])
    result_Avg_Energy.append(temp1)
    result_Mg.append(temp2)
    result_Hc.append(temp3)
    result_Sus.append(temp4)
    result_matrix.append(temp5)
```

Fig 3.6. Two-dimensional Ising model Python code - part6 (calculate properties from 0.4 to 5)

```
j = 0
y_label = ['Average Energy', 'Susceptibility', 'Heat Capacity', 'Magnetization']
for i in [result_Avg_Energy, result_Sus, result_Hc, result_Mg]:
    plt.scatter(T, i)
    plt.xlabel('Temperature')
    plt.ylabel(y_label[j])
    plt.show()
    j += 1
```

Fig 3.7. Two-dimensional Ising model Python code - part7 (graph)

3. Result

-1. Figure

(우 하향으로 1에서부터 0.2씩, 4.8까지 커진다)

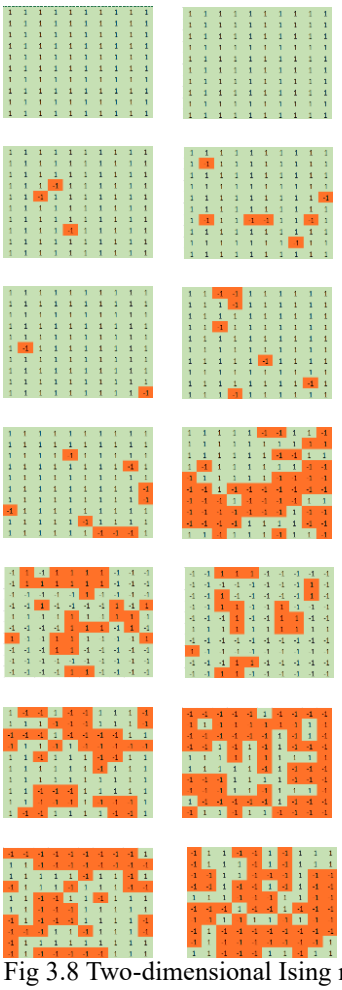


Fig 3.8 Two-dimensional Ising model figure result

위의 figure를 설명하기 전, 위의 그림에서 색깔은 1 spin과 -1 spin을 구분하기 위해서 넣은 것이다. Ising model에서는 +1과 -1의 절대적인 양이 아닌, 상대적인 분포가 중요함으로 결과를 보기 쉽게 하기 위하여 온도가 낮을 때, cell의 개수가 절반이 넘는 종류의 스핀을 초록색으로, 그 반대를 주황색으로 칠하였다.

결과를 살펴보면, 낮은 온도에서는 거의 초록색으로 주변과 동일한 모습을 볼 수 있다. 온도가 올라감에 따라 주변과 상이한 spin들이 나타나기 시작하며 온도가 올라감에 따라 증가한다. 그러다가 약 2.8에서 많이 혼재 되어있는 모습을 보이고 그 이후로 무작위도가 감소하였다가, 다시 온도가 올라가면서 증가하는 그래프를 보인다.

-2. Graph

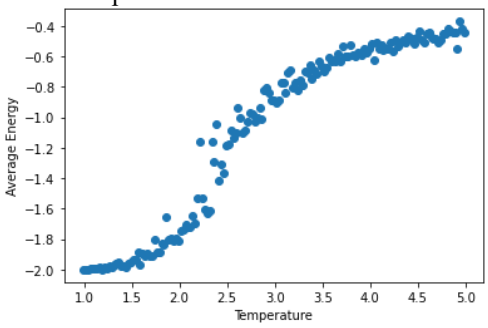


Fig 3.9 Numerical Average Energy graph in the two-dimensional Ising model

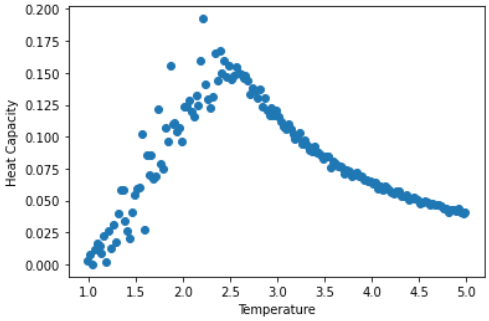


Fig 3.10. Numerical Heat Capacity graph in the two-dimensional Ising model

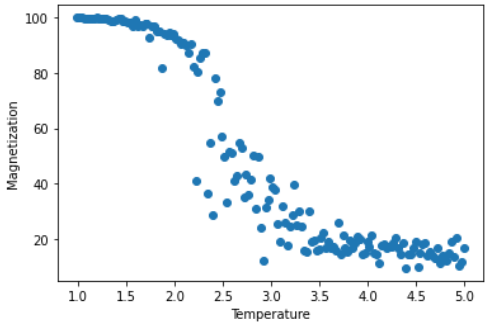


Fig 3.11. Numerical Magnetization graph in the two-dimensional Ising model

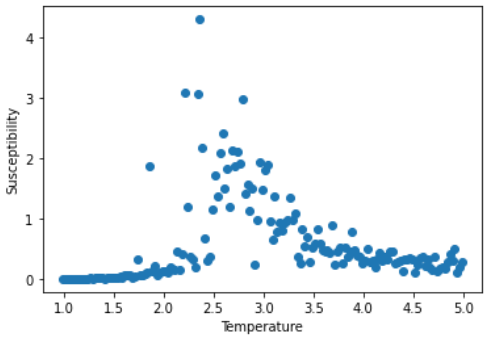


Fig 3.12. Numerical Susceptibility graph in the two-dimensional Ising model

Magnetization, Susceptibility 그래프 모두 약 2.5에서 상전이를 보인다.(Phase Transition)

One-dimensional Ising model에서는 상전이를 관찰할 수 없었기에, 실제 Magnetization을 기술할 수 없었지만, two-dimensional Ising model에서는 상전이를 관찰할 수 있었으므로 이 Model로 이 실제 Magnetization을 가지는 물질을 설명할 수 있음을 확인하였다.

Ⅳ. 검토

1. 결론

하계고체물리인턴을 통하여 Ising Model에 대하여 탐구하여 보았다. 먼저 One-dimensional Ising model에서 수학적인 증명을 통해 Analytical Solution을 구하였다. 이후 Numerical Solution을 구하기 위해 Monte Carlo, Markov Chain, Metropolis Algorithm 등을 탐구하고, 이를 이용하여 One-dimensional Ising Model의 Numerical Solution을 구해보았다. 이 둘을 비교하여 Numerical Solution의 오차에 대해 탐구해보았다.
똑 같은 원리로, Two-dimensional Ising model을 python으로 구현하여 Numerical Solution을 구하였고, one-dimensional Ising Model에서 설명하지 못하였던 상전이가 Two-dimensional Ising model에서는 해결됨을 알고 상전이가 이루어지는 Critical temperature에 대해 살펴보았다.

위 과정을 통해 고체물리에서 Ising 모델과 여러 통계적인 기법에 대해 알게 되고,시뮬레이션을 통해서 고체의 특성을 예측하는 방법에 대해 간단하게 알게 되었다.

2. 제언

위 과정은 이미 약 50년 전에 증명되었다. 50년 전에 이미 밝혀진 fact와 증명 방법을 따라 가는 데에 어려움을 겪었다는 사실에 약간의 무력감을 느꼈으나, 아직 Three-dimensional Ising model의 Analytical solution도 구해지지 않았다는 사실에 놀라며, 이를 증명한 결과만 보았을 때는 쉽지만, 이를 떠올리면서 완전히 이해하는데 시간이 필요함을 당연하게 받아들이게 되었다. 또한 이를 증명한 Ising과 Onsager에게 경외심을 느끼게 되었다.

Reference들을 찾아보며 Ising model이 물리 분야뿐만 아니라 사회 공학적으로 이용될 수 있다고 하였는데, 추후 이 탐구를 심화한다면, 원리를 공부했으니, 이를 실제 사회현상을 예측하는데 이용할 수 있으리라 생각한다.

Ⅴ. Reference

1. Daijiro Yoshioka: Statistical Physics (Springer, 2007)
2. David P.Landau & Kurt Binder: A guide to Monte Carlo Simulation in Statistical Physics Second Eddition (Cambridge, 2005)
3. Giuseppe Grosso, Giuseppe Pastori Parravicini:,Solid State Physics (Second Edition 2014)
4. Werner Krauth: Statistical Mechanics: Algorithms and Computations (Oxford master series in physics, 2006)
5. Siddhartha Chib, Edward Greenburg: Understanding the Metropolis_Hastings Algorithm (The American Statistician, 1995)
6. G Bhanot 1988 Rep. Prog. Phys. 51 429