Monte-carlo Simulation: Ising Model in 2D and 3D

2021 Physics Tripos Submission

May 2021

Abstract

In this project, Monte-carlo simulations of 2D and 3D Ising model are performed based on Metropolis algorithm in Python to investigate critical behaviors of simple ferromagnetic systems. $16^2, 32^2, 64^2$ 2D lattices and 16^3 3D lattice are studied for their decorrelation time, averaged magnetisation, specific heat, magnetic susceptibility, and critical components. Finite size scaling is also examined. Independent estimates of critical temperature T_c are obtained along the investigations of physical properties and combined for all the lattices. Simulation results exhibit signature critical behaviors (i.e. critical slow down, symmetry breaking) as expected. Final T_c results for $16^2, 32^2, 64^2, 16^3$ lattices are $2.309 \pm 0.075, 2.434 \pm 0.075, 2.408 \pm 0.075, 4.601 \pm 0.011$. Critical exponents α, β, δ for 32^2 lattice found to be $-0.12 \pm 0.07, 0.45 \pm 0.26, 33.4 \pm 2.3$. To further improve the simulation, more advanced algorithm and further vectorization are required to boost statistics and reach larger lattice size. (Word count: 2968)

Contents

1 Introduction

Ising model is a simple model that describes phase transition and critical behaviors of a ferromagnetic system. An infinite lattice, coupled to a heat bath, is set up in arbitrary integer dimension (i.e. 2D, 3D) with one spin on each lattice site, as illustrated in Figure 1.

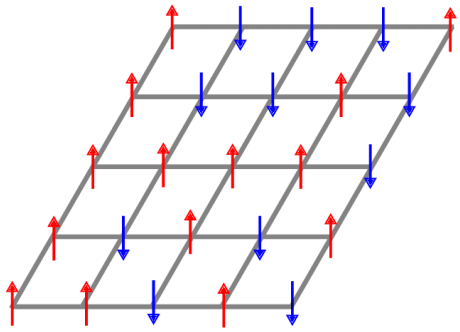

Figure 1: Schematic illustration of 2D Ising model, reprinted from [5]

Energy of the system is

$$E = -J \sum_{\langle ij \rangle} s_i s_j - \mu H \sum_i s_i$$

where s_i is the i^{th} spin, $\langle ij \rangle$ summing over nearest neighbors, J the interaction energy between neighboring spins, μ magnetic moment and H external field. The negative sign suggests a preference of spin alignment, but the coupling to heat bath gives a possibility of $e^{-\frac{\Delta E}{k_B T}}$ for anti-alignment, where ΔE is the energy difference between alignment and anti-alignment state. Ising model predicts alignment below critical temperature T_c and no overall alignment above T_c. [2]

To study critical behaviors, the earliest attempt is to use mean field theory (MFT) where one can implicitly solve $\langle s \rangle$, average spin, from the energy expression, giving

$$\langle s \rangle = \tanh\left(zJ\langle s \rangle / k_B T\right)$$

where z is no. of nearest neighbor. [2] For 2D case, $z = 4$. One can be solved by python root finder and a plot of $\langle s \rangle$ vs temperature is given in Figure 2. Writing temperature in unit of $\frac{J}{k_B T}$, one can easily read from the graph that $T_c = 4$, which is incorrect - 2D analytic solution is $T_c = \frac{2}{ln(1+\sqrt{2})}$, ≈ 2.269. Although we do see the right aligning behavior qualitatively below and above T_c, the quantitative detail is wrong. This result reveals the limitation of MFT, inevitably leading us to numerical simulation for quantitative understanding. Therefore, this project is proposed to illustrate how computational power helps physics understanding, to explore its potential and limitation.

In this project, 2D and 3D Ising models are simulated over small lattices with Metropolis algorithm and periodic boundary condition explained in [1] and lectures. All tasks in [1] are

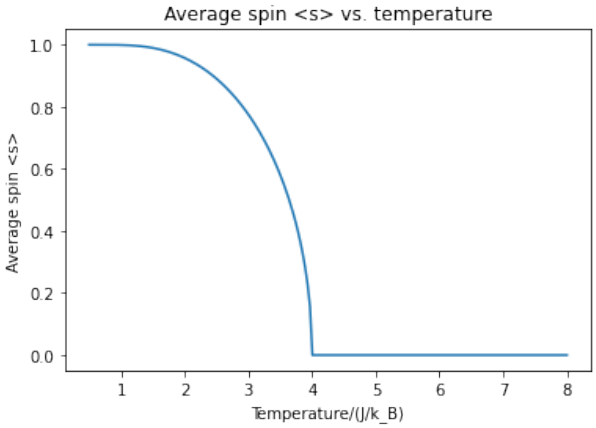

Figure 2: Mean field theory solution: average spin vs temperature

covered but grouped differently and supplemented to better illustrate the reasoning behind this project. In the rest of this report, temperature is given in the unit of $\frac{J}{k_B}$. Section 2 establishes the scope of this project through analysis. Section 3 explains the implementation and performance of this project. Section 4 and 5 present 2D and 3D results respectively, with Section 6 discussing challenges and potential improvements.

2 Analysis

Ising model, as explained in Section 1, describes the phase transition and critical behaviors of a ferromagnetic system at its critical temperature, under the assumption of an infinite lattice and system equilibrium. The model falls under a broader picture of general phase transitions, where transitions are classified as different universal classes characterised by their critical components. Therefore, for a lattice of size $N \times N$, one may be interested in:

1. Finding the **critical temperature**, T_c, since this marks the point of significant changes. T_c can be found by plotting averaged total magnetisation per spin, $\frac{\langle M \rangle}{N^2}$, against temperature in equilibrium state. Ising model predicts a non-zero $\frac{\langle M \rangle}{N^2}$ below T_c and zero otherwise. Therefore, T_c is equal to the temperature where $\frac{\langle M \rangle}{N^2}$ first drops to 0.

2. Investigating the critical behaviors of the system - either thermal or magnetic in nature, manifesting themselves as divergence of **specific heat, C,** and **magnetic susceptibility, χ.** Both are given by fluctuation-dissipation theorem:

$$C = \frac{(\Delta E)^2}{T^2}, \qquad \chi = \frac{(\Delta M)^2}{T}$$

   where $(\Delta E)^2$ and $(\Delta M)^2$ are the variance of total energy and magnetisation. Therefore, one can plot C per spin and χ per spin against T to observe if the critical divergence peak occurs. Also, the position of peaks give an independent estimation of critical temperature, which can be used to verify the result from 1.

3. Computing **critical exponents**, since they are the parameters that characterize a universal class. The critical exponents α, β, δ are given as:

$$C \propto (T_c - T)^{-\alpha}, \qquad M \propto (T_c - T)^{\beta}, \qquad M \propto H^{\frac{1}{\delta}}$$

   for T close to and below T_c. Only in the last relation is H turned on. One can run linear regression with C and M data to find the critical exponents.

For all the topics of interests listed above, numerical result is obtained and compared to analytical solutions in infinite lattice limit. For a 2D system, the analytic solutions are tabulated in Table 1:

Physical Quantity	Analytic Prediction as $N \longrightarrow \infty$
T_c	$\frac{2}{ln(1+\sqrt{2})} (\approx 2.269)$
α	0
β	$\frac{1}{8}$
δ	15

Table 1: Analytical solutions for infinite 2D lattice.

Since Ising model assumes equilibrium state and infinite lattice size, working with finite lattice simulations raises two concerns:

1. **Equilibrium time,** t_e - one needs to know how many Monte-carlo (MC) time steps are required for the system to reach equilibrium, and all the investigation should be based on data coming after t_e. t_e can be either roughly estimated by computing how many steps are required for magnetisation per spin to first fall within 1σ from its equilibrium value, or more quantitatively estimated by calculating decorrelation time,τ_e (see Section 3). Once found, t_e or τ_e determines the total MC steps required - it shall be much larger than t_e and τ_e , but within an acceptable numerical cost (having a sensible runtime). Besides, the phenomenon of **critical slow down** is also expected: t_e and τ_e peaking at T_c, due to the large statistical fluctuation near critical point.

2. **Finite size scaling** - This describes how the finite size of lattice affects the result. One direct consequence of a finite lattice is that, unlike an infinite lattice, C and χ do NOT diverge at T_c - they peak instead. Finite size scaling can be formulated in a variety of ways, among which two are presented:

$$T_C(N) = T_C(\infty) + aN^{-1/v} \tag{1}$$

$$C_{\max}/N^2 \sim \log N \tag{2}$$

where a and are constants. C_max is the specific heat at critical temperature. They can both be examined with simulation data and linear regression. To perform linear regression, ν is assumed to its analytical value: $\nu = 1$. [2]

The nature of equilibrium time problem means the relevant investigations must be conducted before simulating the equilibrium scenarios. Therefore, it is studied first to guide the rest of the project.

The full simulation is only performed for 2D lattice. For 3D lattice, only some of the main properties are studied to form a comparison with [4] for illustrative purpose.

3 Implementation

3.1 2D lattice

Lattices mainly involved in this study are with size $16^2, 32^2, 64^2$. The complete analysis of 2D lattice includes:

1. Determine t_e and τ_e. Study how they vary with **lattice size** and **temperature**.

   t_e, a rough estimate, is given by finding the number of MC time steps required for M per spin to first fall within 1σ from equilibrium state. The entire lattice is swept for 150 MC steps, then M per spin is plotted against MC time for temperature between 0.5 - 10. 8 independent estimates are performed at each temperature. Their average and standard deviation are reported as the equilibrium time estimate and associated error. This analysis is repeated for all three lattice sizes.

   The quantitative estimate is based on autocovariance and decorrelation time. Autocovariance is defined as

   $$A(\tau) = \langle M'(t)M'(t+\tau)\rangle$$

   where $\langle\rangle$ means 'average over a long MC time', and

   $$M' = M - \langle M\rangle$$

   Define autocorrelation as

   $$a(\tau) = A(\tau)/A(0)$$

   and decorrelation time, τ_e, as the time at which

   $$a(\tau_e) = \frac{1}{e}$$

   The definition of tau_e is self-explanatory - it tells how long the system needs to become independent from its initial setting, that is, the time required to 'settle down' in equilibrium. For temperature between 1.5 - 3.5 unit, 8 independent estimates of τ_e are performed at each temperature over 300 MC time. Their average and standard deviation are reported as the τ_e result and associated error. This analysis is repeated for all the three lattice sizes.

   Notice that the different temperature ranges in the above two routines are proposed intentionally: the rough estimate t_e pictures the overall trend of temperature dependence, while the quantitative estimate τ_e pins the position of the peak to prevent underestimation of equilibrium time. An overall result is given by combining the two results for further guidance.

2. Estimate critical temperature.

Magnetisation per spin is recorded over 200 MC time and $\langle M \rangle$ per spin is found. For each temperature between 0.5 - 5 unit, 8 such independent estimates are obtained and averaged. $\langle M \rangle$ per spin is plotted against temperature to read off T_c. This analysis is performed for all three lattice sizes for comparison.

3. Simulate thermal/magnetic critical behaviors.
   Study how C per spin and χ per spin varies with temperature.
   C per spin and χ per spin are evaluated as stated in Section 2 for temperature between 1.5 - 3.5 unit over 300 MC steps on all three lattices. For each temperature, 5 independent simulations are performed and averaged. Then the averaged C per spin and χ per spin are plotted against temperature. Critical temperature is found by reading the position of peaks. The temperature range is deliberately set so to achieve a finer bin size hence higher accuracy of the critical temperature. This analysis improves the precision of critical temperature result based on the previous rough estimate and exemplifies critical behaviors.

4. Find critical exponents.

   The methodology of this section is fairly straightforward - one just find the critical exponents by taking logarithm and perform linear regression. 32^2 lattice is used. See the details in Section 4.

5. Finite size scaling.

   The two statements in Section 2 are studied using linear regression. Lattice size N is varied between 5 and 65, while temperature is varied between 2.15 - 2.35 unit (using an extremely narrow window to precisely probe the position of peak). For each temperature, 3 independent trials are conducted and averaged over 300 MC time.

3.2 3D lattice

To extend the code to 3D scenario, one adds two more nearest neighbors. 16^3 lattice is used. The simulations performed are:

1. Decorrelation time evaluation.
2. Critical temperature estimation.
3. Critical behaviors simulation (C and χ)

Performance To maximize the efficiency and readability, all the physics details are encoded in functions that only take configuration parameters as input argument. This allows the same

functions to be easily adapted for different tasks, and users who do not know all the background physics can also make use of them - they only need to know what parameters are required. The codes are vectorized as much as possible. Performance of tasks are summarised in Table 2:

Task	Typical runtime/min (Google Colab)
2D 16^2	0.83
2D 32^2	5
2D 64^2	20
α, β, δ	5
finite size scaling	15
3D τ_e	20
3D T_c	1
3D C, χ	5

Table 2: Typical runtime for tasks. The 2D tasks refer to tasks performed from Section 4.1 to 4.3 - due to the similar code structure, they share similar run time.

4 2D Lattice Results

4.1 Equilibrium Time

4.1.1 Rough Estimate,t_e

t_e is computed for different lattice sizes, and its temperature dependence is studied. One would expect a sharp peak at T_c where the statistical fluctuation is significant and otherwise constant. (t_e) vs. temperature plots for $16^2, 32^2, 64^2$ lattices are given in Figure 3. The reported estimation for (t_e) and associated error is tabulated.

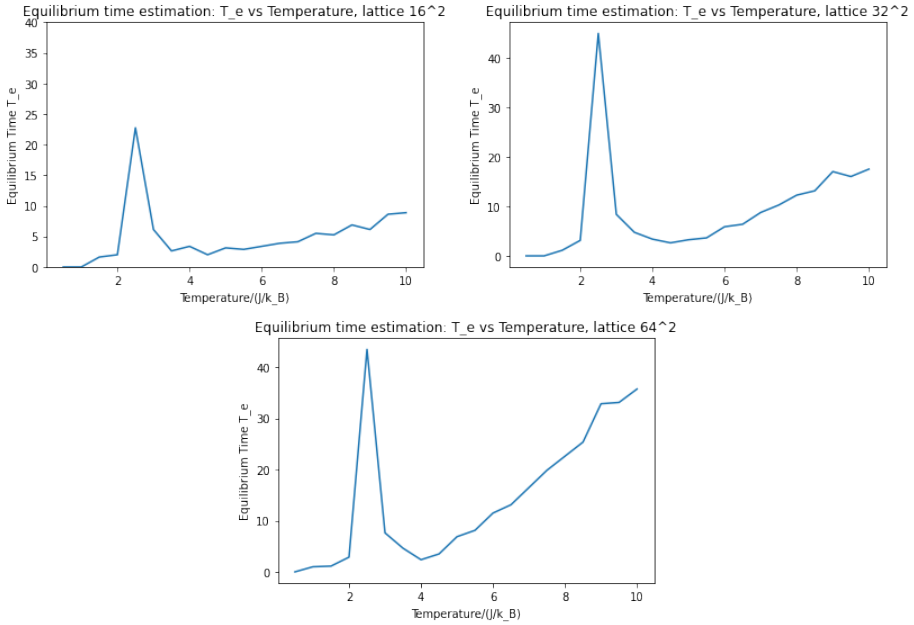

Figure 3: Equilibrium time, as defined in Section 2, is computed and plotted against temperature for lattice$16^2, 32^2, 64^2$.

Lattice size	Equilibrium Time t_e (% frac.)
16^2	20.7±7.7 (37.2%)
32^2	41.1±14.7 (35.6%)
64^2	15.9±6.5 (40.9%)

Table 3: Equilibrium time reported for the three lattices.

The overall temperature dependence of t_e is as expected, sharply peaking at critical temperature. From the table, one may approximate the equilibrium time to be at order $\approx O(50)$. Several features are noticeable:

1. All of the three lattices have t_e peaking at temperature around 2.5, which is close to Onsager's analytic result for critical temperature of an infinite lattice $T_c \approx 2.2692$.

2. The three estimated t_e are all within 1σ from each other, showing consistency. However, all of the results have statistical uncertainty comparable to their values, which suggests that this analysis is merely a rough estimate that serves no more than establishing the order of magnitude for equilibrium time.

3. A peculiar rise in t_e is seen above critical temperature, and the slope appears to increase with lattice size. This is most likely a lattice artifact that is related to both temperature and lattice size. To investigate the nature of this artifact, magnetisation per spin is plotted against MC time and the following comparisons are made based on controlling variables:

    - Comparison between *below* and *above* T_c, lattice size held constant.
    - Comparison between different lattice sizes, temperature held constant above critical temperature.

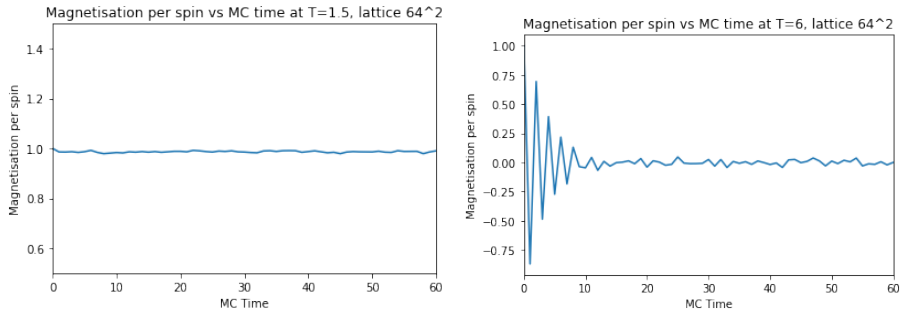

Figure 4: Lattice artifact investigation: M vs MC time at $T = 1.5$ and $T = 6$, while lattice size held constant $= 64^2$

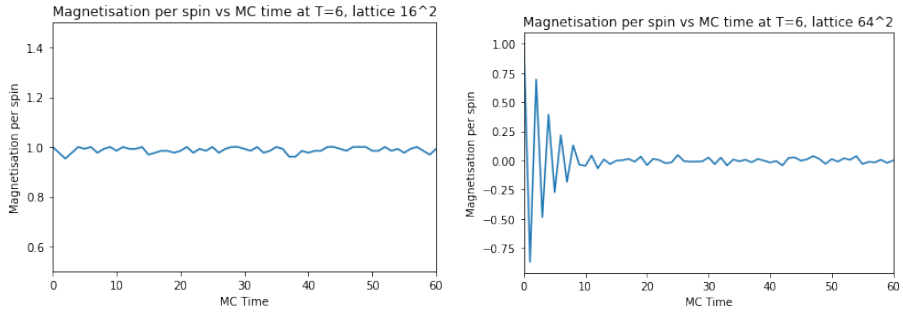

Figure 5: Lattice artifact investigation: M vs MC time with lattice size 16^2 and 64^2, while temperature held constant at $T = 6$.

Figure 4 indicates that the rise in t_e above critical temperature is due to oscillation in magnetisation per spin, which is more likely to occur at high temperature where the system has enough energy to drive upon. Figure 5 verifies the trend seen in Figure 3, where t_e grows more rapidly as the lattice size increases.

In brief, this analysis semi-quantitatively illustrates the phenomenon of critical slow down. It suggests a rough estimate for equilibrium time at $O(50)$, and the unexpected rising beyond T_c might suggest that larger lattice takes more time to reach equilibrium - which is sensible physically since more spins are involved.

4.1.2 Decorrelation Time, τ_e

τ_e is evaluated as a quantitative supplement and T_c is estimated. Results are obtained for all three lattices:

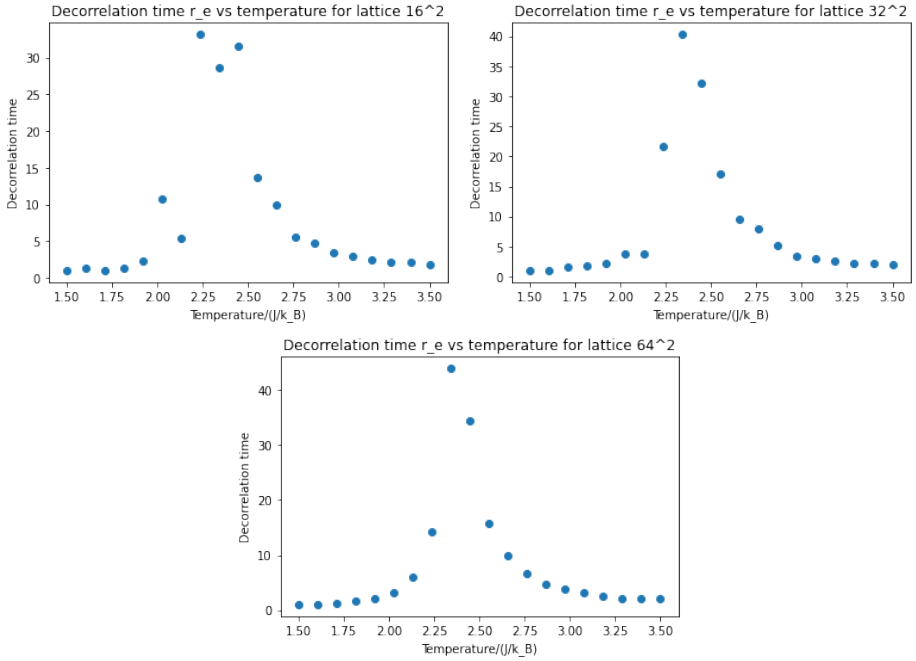

Figure 6: Decorrelation time is computed and plotted against temperature for lattice$16^2, 32^2, 64^2$.

Lattice size	Decorrelation Time τ_e (% frac.)	Critical temperature T_c
16^2	33.1±9.9 (29.9%)	2.236±0.105
32^2	40.3±8.3 (20.6%)	2.342±0.105
64^2	43.8±4.4 (10.0%)	2.342±0.105

Table 4: Reported decorrelation time and critical temperature for lattice $16^2, 32^2, 64^2$. Fractional errors are included in the bracket.

The decorrelation time τ_e results display the same critical peaks as equilibrium time t_e, a sign of critical slow down. Data suggests that:

1. Critical peak for 16^2 lattice is less well-defined than the larger lattices, indicating a larger

statistical uncertainty on each data point. This is a manifestation of finite size effect, and the compromise of accuracy must be acknowledged.

2. The decorrelation times are at $O(50)$ as well, consistent with the previous conclusion for t_e. However, the fractional errors in τ_e are 10% smaller than those in t_e, suggesting τ_e a better quantitative indicator. Besides, extending the temperature to far beyond T_c, τ_e stabilizes rather than rising like t_e:

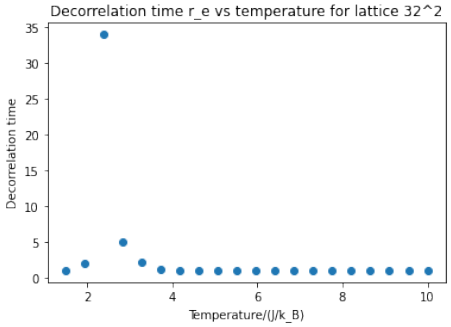

Figure 7: Decorrelation time vs temperature, extended beyond T_c.
The peculiar rise in τ_e is not seen.

3. The reported T_c values are consistent with each other, and they are all within 1σ from Onsager's analytical result for infinite lattice, validating the simulation. Critical temperature appears to increase with lattice size, nevertheless it is not conclusive yet at this stage where only three lattice sizes are studied.

From the investigation of t_e and τ_e, it is concluded that:

1. Time required for the system to reach equilibrium is at order $O(50)$. Therefore, in the subsequent studies lattices are either swept for 200 or 300 MC time steps to equilibrate the system under allowed computational time. Equilibrium state quantities such as magnetisation/energy per spin are computed only from data after 100 MC steps.

2. Critical temperature of lattice with total number of spin at $O(10^2)$ (lattice 16^2) and $O(10^3)$ (lattice $32^2, 64^2$) agrees with Onsager's result within a deviation of $O(10^{-1})$. It validates the simulation, but also foreshadows the high precision required to study finite size scaling.

4.2 Critical Temperature

An independent estimation of T_c is obtained by finding the temperature where $\frac{\langle M \rangle}{N^2}$ at equilibrium goes to zero. Equilibrium state magnetisation per spin is plotted against temperature for all lattice sizes, and T_c results are tabulated:

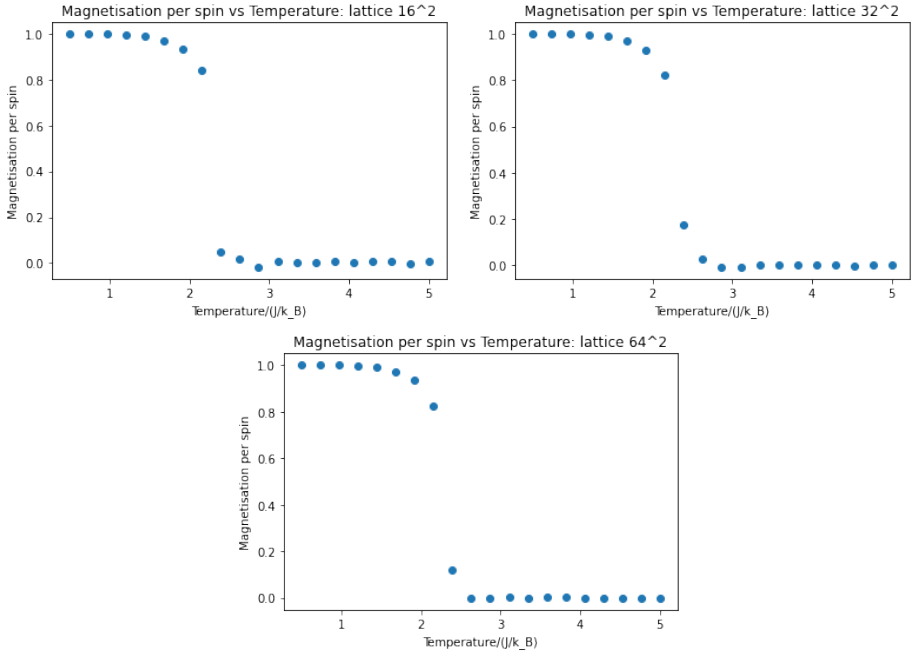

Figure 8: Magnetisation per spin ((averaged in equilibrium state) is computed and plotted against temperature for lattice $16^2, 32^2, 64^2$.

Lattice size	Critical temperature T_c
16^2	2.395±0.237
32^2	2.632±0.237
64^2	2.632±0.237

Table 5: Critical temperature for lattice $16^2, 32^2, 64^2$.

The plots show trend as expected from Ising Model: Averaged magnetisation per spin, $\frac{\langle M \rangle}{N^2}$, is non-zero below T_c due to symmetry breaking and varnishes above T_c. The critical temperatures obtained for three lattices are in agreement with each other within 1σ, and they are consistent with the previous results. Behaviors above critical temperature is stable with no peculiar rise observed since $\langle M \rangle$ is evaluated strictly in non-oscillatory region.

4.3 Specific Heat and Magnetic Susceptibility

Since specific heat is calculated from the variance of energy, it is important to verify the computation of energy is correct. For this purpose, energy per spin is computed for temperature in $[0.5, 5]$ for 32^2 lattice. Energy per spin vs. temperature plot is made and compared to [2] in Figure 9.

As explained in [2], Ising model predicts energy per spin below T_c to be -2 when temperature is represented in unit of J/k_B, and rises to 0 at high temperature since spin will be randomly

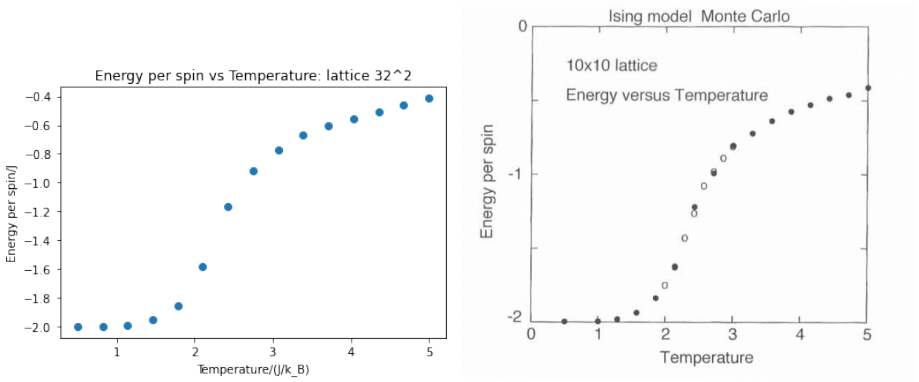

Figure 9: (left) Energy per spin vs temperature for 32^2 lattice, obtained in this project. (right) Energy per spin vs temperature for 10^2 lattice, reprinted from [2].

oriented. The plot obtained in this project is consistent with the one in [2]: both show the correct energy of -2 below T_c, rising to -0.4 at $T = 5$ which is far from 0. The deviation from 0 in high temperature limit indicates orientation of spins is not random, even when the magnetisation averaged to zero. This correlation in orientation among spins turns out to be an important subject of interests. (See [2])

As energy evaluation is validated, temperature dependence of specific heat per spin and magnetic susceptibility per spin for all three lattices is plotted in Figure 10, 11 with T_c obtained from both quantities tabulated in Table 6:

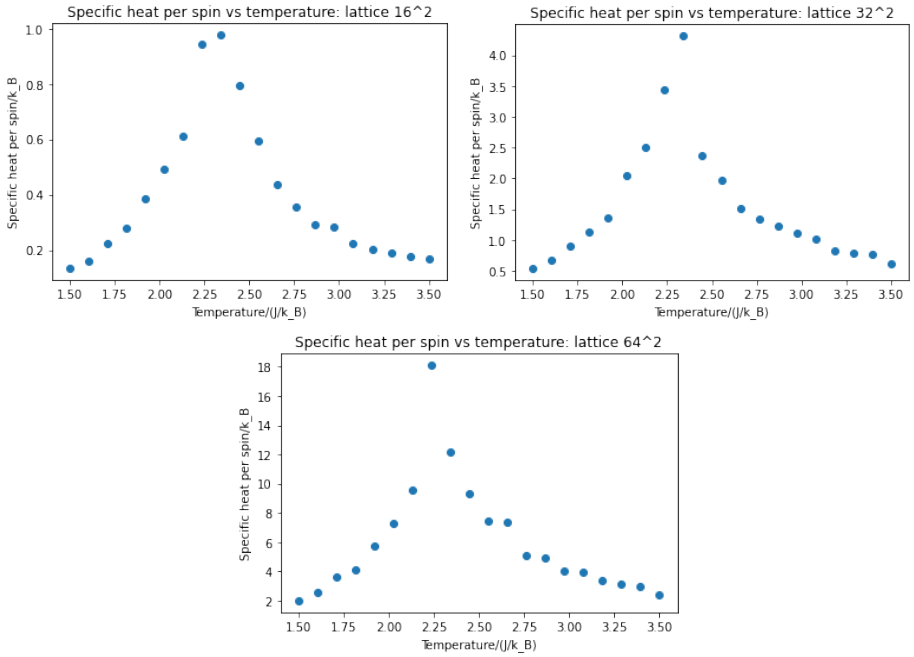

Figure 10: Specific heat per spin vs temperature for lattice $16^2, 32^2, 64^2$.

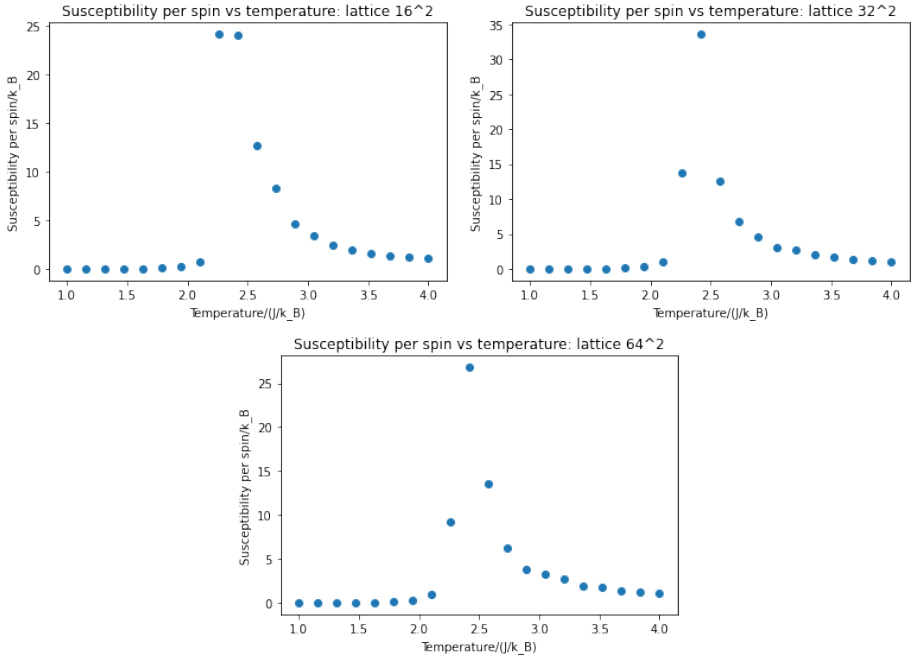

Figure 11: Magnetic susceptibility per spin vs temperature for lattice $16^2, 32^2, 64^2$.

Lattice size	Specific Heat T_c	Magnetic Susceptibility T_c
16^2	2.342±0.105	2.263±0.105
32^2	2.342±0.105	2.421±0.105
64^2	2.237±0.105	2.421±0.105

Table 6: T_c estimation from specific heat and magnetic susceptibility for lattice $16^2, 32^2, 64^2$.

Features of data include:

1. The temperature dependence of C per spin and χ per spin agrees with Ising model, showing critical divergence. Peak width appears to have a weak dependence on lattice size. However, susceptibility peak are sharper than specific heat peak.

2. A subtle disagreement in lattice size dependence of T_c can be observed from both the plots and the tabulated data. In Figure 10, C peak shifts leftward as lattice size increases while in Figure 11 χ peak shifts rightward, which is confirmed by Table 6 where specific heat T_c estimation decreases as lattice size grows while magnetic susceptibility T_c estimation shows an opposite trend. With only three lattices and limited data resolution, one cannot yet reach a conclusive claim. This will be discussed in Section 6.

3. Results in Table 4 are consistent with each other and previous results within 1σ.

4.4 Critical Exponents

4.4.1 Evaluation of α

Since α is expected to be 0 for infinite lattice [2], specific heat shall be a constant and doesn't depend on reduced temperature $t = T_c - T$ near T_c. Therefore, specific heat is plotted against reduced temperature t for 32^2 lattice close to T_c, as shown below:

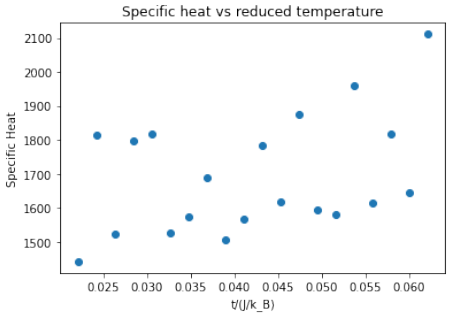

Figure 12: Specific heat vs reduced temperature t for 32^2 lattice.

Figure 12 indeed shows a constant trend with a large scattering about $C \approx 1700$. Linear regression is performed between ln(specific heat) and ln(t). α is found to be -0.12 ± 0.07, showing a 2σ difference from that of an infinite lattice. This is a manifestation of finite size scaling.

4.4.2 Evaluation of β

β is expected to be 0.125 for infinite lattice. To obtain β, one plots ln(Magnetisation) against ln(t) and performs linear regression, shown in Figure 13:

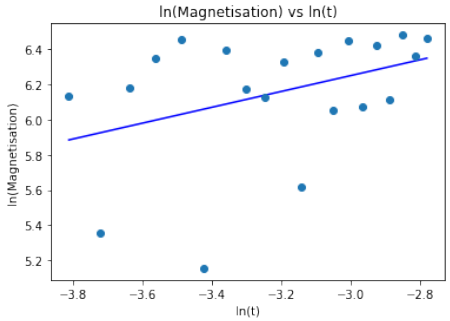

Figure 13: ln(Magnetisation) vs ln(reduced temperature) for 32^2 lattice.

A general linear trend can be seen despite several outliers. Linear regression reports β value of

0.45 ± 0.26, suspending a 1.2σ difference from that of infinite lattice, another sign of finite size effect.

4.4.3 Evaluation of δ

δ is the critical component that links magnetisation of the ferromagnetic system to external magnetic field. Such dependence manifests itself as 'hysteresis': contrary to other magnetic systems where magnetisation varnishes when external field is off, ferromagnet has a remnant magnetisation at the absence of the field below T_c. Before computing δ, such phenomenon is first simulated and observed. $\langle M \rangle$ per spin is plotted against external field for temperature below and above T_c on 32^2 lattice, shown as below:

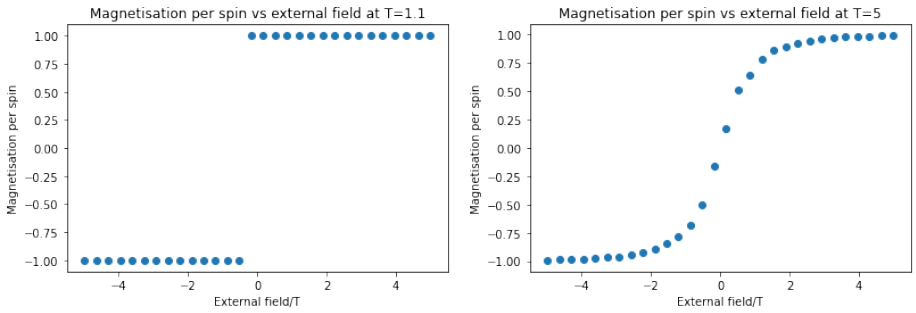

Figure 14: Hysteresis: $\langle M \rangle$ per spin vs external field strength on 32^2 lattice for T=1.1 (below T_c) and T= 5 (above T_c)

Hysteresis is observed as expected. Toward the two ends of plot, magnetisation saturates as the spins fully align. At $H = 0$, above T_c magnetisation per spin varnishes, while below T_c a discontinuous jump is seen, indicating remnant magnetisation.

δ is expected to be 15 for infinite lattice. At T_c (≈ 2.34 for 32^2 lattice), δ is evaluated by performing linear regression on ln(M) and ln(H). Result is shown Figure 15:

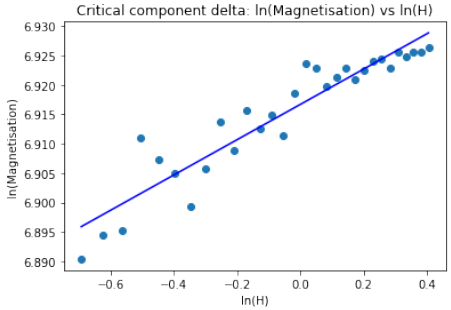

Figure 15: Critical exponent δ: ln(Magnetisation) vs ln(external field) with fitting.

From Figure 15, the linearity is evident. Reported δ is 33.4 ± 2.3, significantly deviating from that of infinite lattice.

4.5 Finite Size Scaling

4.5.1 Statement 1

As explained in Section 2, Equation (1) is examined with simulation. Assuming $\nu = 1$, T_c is plotted against $\frac{1}{N}$ and linear regression is performed, shown in Figure 16:

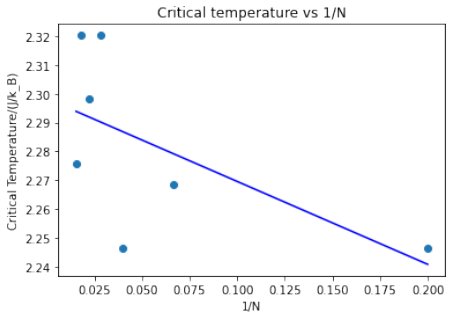

Figure 16: Finite size scaling: Critical temperature vs $\frac{1}{N}$, best fit line is added onto data points.

On the plot, data falls in a rough linear trend with large scattering about the fitted line, suggesting large statistical uncertainties. This is confirmed by the linear regression with r value reported as -0.602. Numerical result reported gives $T_c(\infty) = 2.298 \pm 0.171$, consistent with the expected analytic value $T_c(\infty) \approx 2.269$.

4.5.2 Statement 2

Equation (2) is examined by simulation, and a linear regression is performed between specific heat per spin at T_c and ln (Lattice size, N), shown in Figure 17. The slope is expected to be 1.

A moderate linearity is observed with data scattering about the linear fit, r value reported to be 0.677. Slope is reported to be 0.284 ± 0.138, deviating significantly from expected value (=1). This consistency might be due to low sample size. Potential remedy is discussed in Section 6.

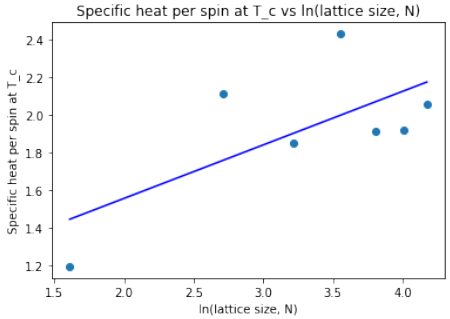

Figure 17: Finite size scaling: Specific heat per spin at T_c vs ln(Lattice size, N), best fit line is added onto data points.

5 3D Lattice Results

5.1 Decorrelation Time

Decorrelation time is evaluated on 16^3 lattice similarly as in Section 4.1.2. Result is shown below:

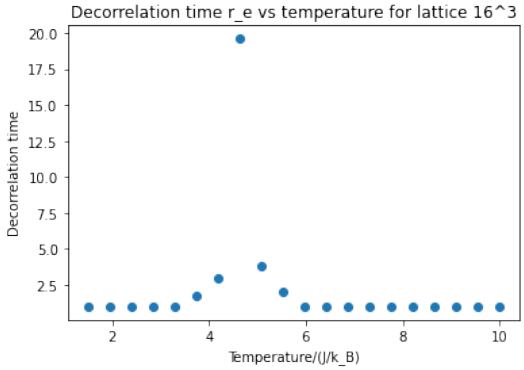

Figure 18: Decorrelation time vs temperature for 16^3 lattice.

Critical slow down is observed as expected. It is concluded that the decorrelation time is at $O(20)$, hence 200 - 300 MC time steps are sufficient to set the system in equilibrium.

5.2 Critical Temperature

Magnetisation per spin $\frac{\langle M \rangle}{N^2}$ at equilibrium state is plotted against temperature to estimate T_c. Result is compared to a recent study [4], as shown in Figure 19 :

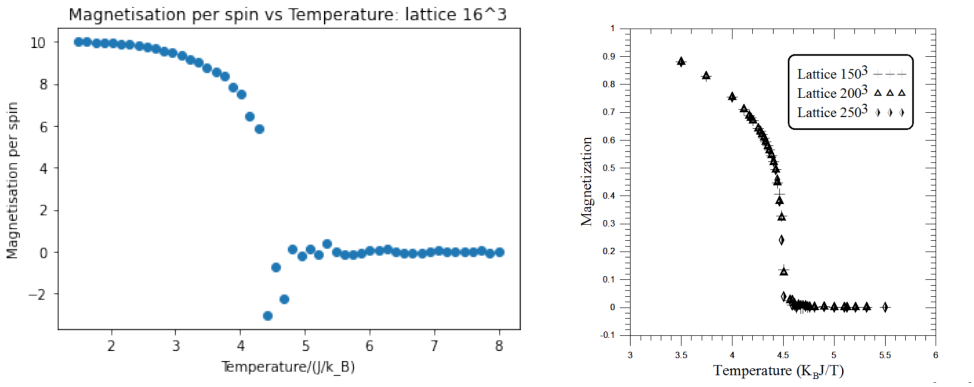

Figure 19: (left) Magnetisation per spin vs temperature at equilibrium state for 16^3 lattice, obtained in this project. (right) Same plot on $150^3, 200^3, 250^3$ lattice obtain in [4], reprinted.

T_c reported in this simulation is 4.816 ± 0.133. On Figure 19, both plots show similar trend that agrees with theoretical prediction, and in both cases magnetisation per spin drops to zero at critical temperature ≈ 4.5.

It is noticeable that 16^3 lattice plot (left) overshoots into negative magnetisation, while larger lattices don't. This can be attributed to a much larger statistical uncertainty for this study: small lattice size contributes to finite size effect, and a relatively small MC time span limits the quality of statistics. Only 200 sweeps are performed in this study, contrary to the quoted study [4] where 10000 sweeps are performed - 50 times larger sample size.)

The result $T_c = 4.816 \pm 0.133$ is compared to a study in 2000 [3] which gives $T_c = 4.515 \pm 0.025$, showing 2σ tension. However, since [3] has additional implementation of Wolff dynamics and the statistics is 5 times larger, discrepancy is possible as an artifact of different computational configurations.

5.3 Specific Heat and Susceptibility

Specific heat per spin and magnetic susceptibility per spin are plotted against temperature for 16^3 lattice and compared to [4] result, shown in Figure 20. Features of plots include:

- All plots show critical slow down as expected. In 3D case, specific heat peak is wider than susceptibility peak just as 2D case discussed in Section 4.3.
- Susceptibility plot obtained in this study is consistent with that from [4], both peaking at $T_c \approx 4.5$.
- An evident asymmetry is observed from the different sides of specific heat peak. Specific heat appears to decay faster above T_c than below T_c. The origin of this asymmetry is likely a lattice artifact and needs to be confirm by further study.
- Specific heat peak reports $T_c = 4.448 \pm 0.103$. Susceptibility peak reports $T_c = 4.538 \pm 0.077$. They agree much better with [3], perfectly within 1σ tension. This potentially suggests that it is more precise to use critical divergence peak for T_c determination than using $\langle M \rangle$.

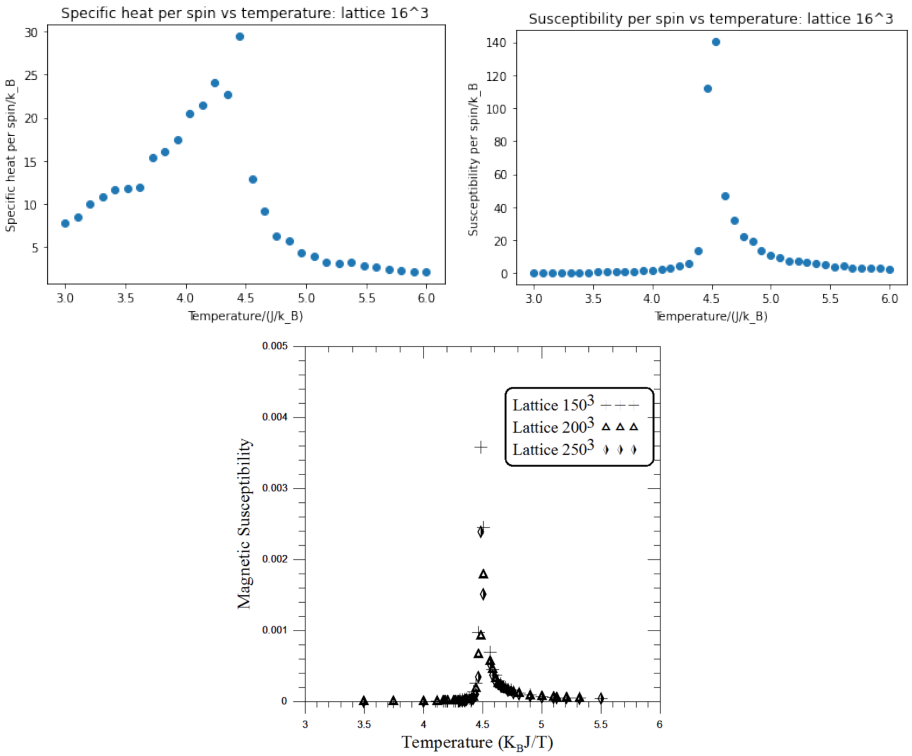

Figure 20: (up left) Specific heat per spin vs temperature at equilibrium state for 16^3 lattice, obtained in this project. (up right) Susceptibility per spin vs temperature at equilibrium state for 16^3 lattice, obtained in this project. (bottom) Susceptibility plot on $150^3, 200^3, 250^3$ lattice obtain in [4], reprinted. Only the magnetisation plot is given by [4].

6 Discussion

Results from Section 5 are summarized in Table 7, 8, and 9 :

Lattice size	Decorrelation Time T_c	Magnetisation T_c	Specific Heat T_c	Magnetic Susceptibility T_c	Combined Result T_c
16^2	2.236±0.105	2.395±0.237	2.342±0.105	2.263±0.105	2.309±0.075
32^2	2.342±0.105	2.632±0.237	2.342±0.105	2.421±0.105	2.434±0.075
64^2	2.342±0.105	2.632±0.237	2.237±0.105	2.421±0.105	2.408±0.075

Table 7: T_c results summary for 2D lattices. A combined result is given for each lattice size by averaging all results obtained with uncertainties added in quadrature.

Lattice size	Magnetisation T_c	Specific Heat T_c	Magnetic Susceptibility T_c	Combined Result T_c
16^2	4.816±0.133	4.448±0.103	4.538±0.077	4.601±0.011

Table 8: T_c results summary for 3D lattices. A combined result is given for lattice 16^3 by averaging all results obtained with uncertainties added in quadrature.

Lattice size	α	β	δ
32^2	-0.12±0.07	0.45±0.26	33.4±2.3
∞	0	0.125	15

Table 9: 2D critical components computed on 32^2 lattice and their expected values for infinite lattice (shown in the bottom row).

The main challenge faced by this study is rooted in the vectorization of codes. Despite heavy efforts to vectorize, a considerable number of loops is still required, bringing up computational cost hence limiting the available number of sweeps (MC time steps), number of independent trials, data spacing and size of lattice. The first three limitations introduces statistical uncertainty exemplified by:

- the wide, fuzzy shape of 2D and 3D specific heat peaks
- significant scattering in α, β, δ regression plots
- significant scattering in both finite size scaling investigations

Since total number of loops performed depends on number of lattice sites, the size of lattice must be compromised to ensure enough sweeps are performed for system equilibrium, giving 64^2 as the limit for 2D case and 16^3 for 3D cases in this project, while ideally lattices at much larger order of magnitude shall also be studied. The limitation in lattice size, along with the other limitations, indicates the following questions cannot be conclusively addressed by this project:

- how τ_e varies with lattice size
- why specific heat peak and χ peak shift oppositely as lattice size increases
- inconsistency in finite size scaling Statement 2 result
- origin of the asymmetry in 3D specific peak

Further improvements to this study shall focus on vectorizing the codes and improving the algorithm. Ideally, lattice size shall be boosted to $128^2, 256^2$ etc, with available MC time steps increased to ≥ 1000 steps, temperature data spacing reduced to $O(10^{-2})$. This shall improve the statistics and bring clarity to the puzzles unsolved yet.

References

[1] David Buscher. 2021.

[2] Nicholas J Giordano and Hisao Nakanishi. In: *Computational Physics*. Pearson Education, 1997, pp. 235–264.

[3] Peter J Meyer. "Computational Studies of Pure and Dilute Spin Models". PhD thesis. Hermetic Systems website, 2000, pp. 19–19.

[4] A F Sonsin et al. "Computational Analysis of 3D Ising Model Using Metropolis Algorithms". In: *Journal of Physics: Conference Series* 630 (July 2015), p. 012057. DOI: 10.1088/1742-6596/630/1/012057. URL: https://doi.org/10.1088/1742-6596/630/1/012057.

[5] Sascha Wald. "Thermalisation and Relaxation of Quantum Systems". PhD thesis. Sept. 2017. DOI: 10.13140/RG.2.2.25169.63842.

(Word count: 2968)

Appendix

The codes are directly exported from Google Colab, hence might be subjected to formatting issue. To run the code, please find the .ipynb file attached on TiS and open it in Google Colab.

```
# -*- coding: utf-8 -*-
"""Ising_final.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/14ShyGxfBBZNBNq-nKggnk9XpqtRyg-sc

# Part 1: MFT breaking down

##Task 1: wrong critical temperature
"""

import numpy as np
import random as rd
from math import tanh,log
import matplotlib.pyplot as plt
from scipy.optimize import fsolve
from scipy import stats

T=np.linspace(0.5,8,200)
S=[]
for t in T:
  def f(s):
    return s-tanh(4*s/t)
  s=fsolve(f,0.5)
  S.append(s)

plt.plot(T,S)
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Average spin <s>')
plt.title('Average spin <s> vs. temperature')
plt.show()

"""#Part 2: 2D lattice
```

```
## Task 1: Equilibrium time

### Rough Estimate

#### 16*16 lattice
"""

import numpy as np
import random as rd
from math import exp,sqrt
import matplotlib.pyplot as plt


#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
```

```
  return M

def findtE(M,MC_Time):
  q=len(M)-30
  M_eq=np.mean(M[q:])
  std=np.std(M[q:])
  for x in range(len(M)):
    if np.abs(M[x]-M_eq)>std:
      pass
    else:
      return x

#16*16

tt=[]
T=np.linspace(0.5,10,num=20)
MC_Time=np.linspace(0,150,num=150,dtype=int)
tt_err=[]
for t in T:
  tE=[]
  for x in range(8):
    LAT=constrct_lattice(16)
    M=[]
    for y in range(150): #find M after 149 sweeps
      M.append(findM(LAT,16))
      LAT=sweep(LAT,16,t,H)
    tE.append(findtE(M,MC_Time))
  tE_f=[x for x in tE if x is not None]
  tt.append(np.mean(tE_f))
  tt_err.append(np.std(tE_f)sqrt(8))
#8 runs and find average
print('Error on T_e at critical temperature
↪  is',tt_err[list(tt).index(np.max(tt))])
print('Error on T_e in this study is at the order of',sqrt(np.sum([x**2 for x
↪  in tt_err])))
plt.plot(T,tt)
plt.title('Equilibrium time estimation: T_e vs Temperature, lattice 16^2')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Equilibrium Time T_e')
plt.ylim([0,40])

"""####32*32 lattice"""
```

```
import numpy as np
import random as rd
from math import exp,sqrt
import matplotlib.pyplot as plt


#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findtE(M,MC_Time):
  q=len(M)-30
  M_eq=np.mean(M[q:])
  std=np.std(M[q:])
  for x in range(len(M)):
```

```
    if np.abs(M[x]-M_eq)>std:
      pass
    else:
      return x

#32*32

tt=[]
T=np.linspace(0.5,10,num=20)
MC_Time=np.linspace(0,150,num=150,dtype=int)
tt_err=[]
for t in T:
  tE=[]
  for x in range(8):
    LAT=constrct_lattice(32)
    M=[]
    for y in range(150): #find M after 149 sweeps
      M.append(findM(LAT,32))
      LAT=sweep(LAT,32,t,H)
    tE.append(findtE(M,MC_Time))
  tE_f=[x for x in tE if x is not None]
  tt.append(np.mean(tE_f))
  tt_err.append(np.std(tE_f)/sqrt(8))
#8 runs and find average
print('Error on T_e at critical temperature
↪  is',tt_err[list(tt).index(np.max(tt))])
print('Error on T_e in this study is at the order of',sqrt(np.sum([x**2 for x
↪  in tt_err])))
plt.plot(T,tt)
plt.title('Equilibrium time estimation: T_e vs Temperature, lattice 32^2')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Equilibrium Time T_e')

"""####64*64 lattice"""

import numpy as np
import random as rd
from math import exp,sqrt
import matplotlib.pyplot as plt


#parameter list
H=0
```

```
#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findtE(M,MC_Time):
  q=len(M)-30
  M_eq=np.mean(M[q:])
  std=np.std(M[q:])
  for x in range(len(M)):
    if np.abs(M[x]-M_eq)>std:
      pass
    else:
      return x

#64*64

tt=[]
```

```
T=np.linspace(0.5,10,num=20)
MC_Time=np.linspace(0,150,num=150,dtype=int)
tt_err=[]
for t in T:
  tE=[]
  for x in range(8):
    LAT=constrct_lattice(64)
    M=[]
    for y in range(150): #find M after 149 sweeps
      M.append(findM(LAT,64))
      LAT=sweep(LAT,64,t,H)
    tE.append(findtE(M,MC_Time))
  tE_f=[x for x in tE if x is not None]
  tt.append(np.mean(tE_f))
  tt_err.append(np.std(tE_f)sqrt(8))
#8 runs and find average
print('Error on T_e at critical temperature
↪  is',tt_err[list(tt).index(np.max(tt))])
print('Error on T_e in this study is at the order of',sqrt(np.sum([x**2 for x
↪  in tt_err])))
plt.plot(T,tt)
plt.title('Equilibrium time estimation: T_e vs Temperature, lattice 64^2')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Equilibrium Time T_e')

"""###Decorrelation time

####16*16 lattice
"""

import numpy as np
import random as rd
from math import exp
import matplotlib.pyplot as plt



#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l
```

```
def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findAcov(M,tau): #find autocorelation
  M_av=np.mean(M)
  M_prime=[x-M_av for x in M]
  t_tot=len(M)
  A=np.mean([M_prime[x]*M_prime[x+tau] for x in range(t_tot-tau)])
  return A

def findtau_e(M):
  A0=findAcov(M,0)
  for t in range(1,len(M)):
    At=findAcov(M,t)
    a=At/A0
    if a>exp(-1):
      pass
    else:
      return t

#16*16 lattice
```

```
T=np.linspace(1.5,3.5,20)
te=np.zeros((8,20))
for i in range(8):
  for j in range(20):
    M=[]
    LAT=constrct_lattice(16)
    for x in range(300):
      LAT=sweep(LAT,16,T[j],H)
      M.append(findM(LAT,16))
    te[i][j]=findtau_e(M)
te_f=np.average(te,axis=0)
te_err=np.std(te,axis=0)
print('Critical temperature is found to be',T[list(te_f).index(np.max(te_f))])
print('Decorrelation time at critical temperature
↪  is',max(te_f),'+-',te_err[list(te_f).index(np.max(te_f))]/sqrt(8))
plt.plot(T,te_f,'o')
plt.title('Decorrelation time r_e vs temperature for lattice 16^2')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Decorrelation time')
plt.show()

"""####32*32 lattice"""

import numpy as np
import random as rd
from math import exp
import matplotlib.pyplot as plt


#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
```

```
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findAcov(M,tau): #find autocorelation
  M_av=np.mean(M)
  M_prime=[x-M_av for x in M]
  t_tot=len(M)
  A=np.mean([M_prime[x]*M_prime[x+tau] for x in range(t_tot-tau)])
  return A

def findtau_e(M):
  A0=findAcov(M,0)
  for t in range(1,len(M)):
    At=findAcov(M,t)
    a=At/A0
    if a>exp(-1):
      pass
    else:
      return t

#32*32 lattice
T=np.linspace(1.5,3.5,20)
te=np.zeros((8,20))
for i in range(8):
  for j in range(20):
    M=[]
    LAT=constrct_lattice(32)
    for x in range(300):
```

```
      LAT=sweep(LAT,32,T[j],H)
      M.append(findM(LAT,32))
    te[i][j]=findtau_e(M)
te_f=np.average(te,axis=0)
te_err=np.std(te,axis=0)
print('Critical temperature is found to be',T[list(te_f).index(np.max(te_f))])
print('Decorrelation time at critical temperature
↪  is',max(te_f),'+-',te_err[list(te_f).index(np.max(te_f))]/sqrt(8))
plt.plot(T,te_f,'o')
plt.title('Decorrelation time r_e vs temperature for lattice 32^2')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Decorrelation time')
plt.show()

"""####64*64 lattice"""

import numpy as np
import random as rd
from math import exp,sqrt
import matplotlib.pyplot as plt



#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
```

```
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findAcov(M,tau): #find autocorelation
  M_av=np.mean(M)
  M_prime=[x-M_av for x in M]
  t_tot=len(M)
  A=np.mean([M_prime[x]*M_prime[x+tau] for x in range(t_tot-tau)])
  return A

def findtau_e(M):
  A0=findAcov(M,0)
  for t in range(1,len(M)):
    At=findAcov(M,t)
    a=At/A0
    if a>exp(-1):
      pass
    else:
      return t

#64*64 lattice
T=np.linspace(1.5,3.5,20)
te=np.zeros((8,20))
for i in range(8):
  for j in range(20):
    M=[]
    LAT=constrct_lattice(64)
    for x in range(300):
      LAT=sweep(LAT,64,T[j],H)
      M.append(findM(LAT,64))
    te[i][j]=findtau_e(M)
te_f=np.average(te,axis=0)
te_err=np.std(te,axis=0)
print('Critical temperature is found to be',T[list(te_f).index(np.max(te_f))])
```

```
print('Decorrelation time at critical temperature
↪  is',max(te_f),'+-',te_err[list(te_f).index(np.max(te_f))]/sqrt(8))
plt.plot(T,te_f,'o')
plt.title('Decorrelation time r_e vs temperature for lattice 64^2')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Decorrelation time')
plt.show()

"""####32*32 long range"""

import numpy as np
import random as rd
from math import exp
import matplotlib.pyplot as plt



#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
```

```
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findAcov(M,tau): #find autocorelation
  M_av=np.mean(M)
  M_prime=[x-M_av for x in M]
  t_tot=len(M)
  A=np.mean([M_prime[x]*M_prime[x+tau] for x in range(t_tot-tau)])
  return A

def findtau_e(M):
  A0=findAcov(M,0)
  for t in range(1,len(M)):
    At=findAcov(M,t)
    a=At/A0
    if a>exp(-1):
      pass
    else:
      return t

#32*32 lattice
T=np.linspace(1.5,10,20)
te=np.zeros((8,20))
for i in range(8):
  for j in range(20):
    M=[]
    LAT=constrct_lattice(32)
    for x in range(300):
      LAT=sweep(LAT,32,T[j],H)
      M.append(findM(LAT,32))
    te[i][j]=findtau_e(M)
te_f=np.average(te,axis=0)
te_err=np.std(te,axis=0)
'''
print('Critical temperature is found to be',T[list(te_f).index(np.max(te_f))])
print('Decorrelation time at critical temperature
↪  is',max(te_f),'+-',te_err[list(te_f).index(np.max(te_f))])
'''
plt.plot(T,te_f,'o')
plt.title('Decorrelation time r_e vs temperature for lattice 32^2')
plt.xlabel('Temperature/(J/k_B)')
```

```
plt.ylabel('Decorrelation time')
plt.show()

"""## Task 2: Critical temperature

###16*16 lattice
"""

import numpy as np
import random as rd
from math import exp
import matplotlib.pyplot as plt


#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
```

```
  M=sum/(N**2)
  return M

#16*16

# Mean magnetisation is equal to sum of total M divided by the number of steps
T=np.linspace(0.5,5,20)
M_av=np.zeros((8,20))
for i in range(8):
  for j in range(20):
    M=[]
    LAT=constrct_lattice(16)
    for x in range(200): #find M after 199 sweeps
      M.append(findM(LAT,16))
      LAT=sweep(LAT,16,T[j],H)
    M_av[i][j]=np.mean(M[100:])
M_f=np.average(M_av,axis=0)
def CT(M_f,T):
  p=np.isclose(M_f,np.zeros(20),atol=0.05)
  id=0
  for i in p:
    if i == False:
      pass
    else:
      id=list(p).index(i)
      break
  return T[id]
print('Critical temperature is found to be',CT(M_f,T))
plt.plot(T,M_f,'o')
plt.title('Magnetisation per spin vs Temperature: lattice 16^2')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Magnetisation per spin')
plt.show()

"""###32*32 lattice"""

import numpy as np
import random as rd
from math import exp
import matplotlib.pyplot as plt


#parameter list
```

```
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum/(N**2)
  return M

#32*32

# Mean magnetisation is equal to sum of total M divided by the number of steps
T=np.linspace(0.5,5,20)
M_av=np.zeros((8,20))
for i in range(8):
  for j in range(20):
    M=[]
    LAT=constrct_lattice(32)
    for x in range(200): #find M after 199 sweeps
      M.append(findM(LAT,32))
      LAT=sweep(LAT,32,T[j],H)
```

```
    M_av[i][j]=np.mean(M[100:])
M_f=np.average(M_av,axis=0)
def CT(M_f,T):
  p=np.isclose(M_f,np.zeros(20),atol=0.05)
  id=0
  for i in p:
    if i == False:
      pass
    else:
      id=list(p).index(i)
      break
  return T[id]
print('Critical temperature is found to be',CT(M_f,T))
plt.plot(T,M_f,'o')
plt.title('Magnetisation per spin vs Temperature: lattice 32^2')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Magnetisation per spin')

"""###64*64 lattice"""

import numpy as np
import random as rd
from math import exp
import matplotlib.pyplot as plt


#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti┘
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
```

```
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum/(N**2)
  return M

#64*64

# Mean magnetisation is equal to sum of total M divided by the number of steps
T=np.linspace(0.5,5,20)
M_av=np.zeros((8,20))
for i in range(8):
  for j in range(20):
    M=[]
    LAT=constrct_lattice(64)
    for x in range(200): #find M after 199 sweeps
      M.append(findM(LAT,64))
      LAT=sweep(LAT,64,T[j],H)
    M_av[i][j]=np.mean(M[100:])
M_f=np.average(M_av,axis=0)
def CT(M_f,T):
  p=np.isclose(M_f,np.zeros(20),atol=0.05)
  id=0
  for i in p:
    if i == False:
      pass
    else:
      id=list(p).index(i)
      break
  return T[id]
print('Critical temperature is found to be',CT(M_f,T))
plt.plot(T,M_f,'o')
plt.title('Magnetisation per spin vs Temperature: lattice 64^2')
plt.xlabel('Temperature/(J/k_B)')
```

```
plt.ylabel('Magnetisation per spin')

"""## Task 3: C and \Chi

### Specific heat

#### Energy computation
"""

import numpy as np
import random as rd
from math import exp
import matplotlib.pyplot as plt


#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
```

```
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findE(lattice,N,H): #find energy per spin of the lattice
  E_field=-H*np.sum(np.sum(lattice,axis=0))
  E_i=0
  for i in range(N):
    for j in range(N):
      E_i+= -lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+lattice
      ↪  [i][(j+1+N)%N]+lattice[i][(j-1+N)%N])
  return (E_field+E_i)/(2*(N**2))

#verify the energy function is correct


T=np.linspace(0.5,5,15)
LAT=constrct_lattice(32)
Ef=[]
for t in T:
  E=[]
  for i in range(300):
    LAT=sweep(LAT,32,t,H)
    E.append(findE(LAT,32,H))
  Ef.append(np.mean(E[200:]))

plt.plot(T,Ef,'o')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Energy per spin/J')
plt.title('Energy per spin vs Temperature: lattice 32^2')
plt.show()

"""####16*16 lattice"""

import numpy as np
import random as rd
from math import exp
import matplotlib.pyplot as plt


#parameter list
H=0
```

```
#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findE(lattice,N,H): #find energy per spin of the lattice
  E_field=-H*np.sum(np.sum(lattice,axis=0))
  E_i=0
  for i in range(N):
    for j in range(N):
      E_i+= -lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+lattice
      ↪  [i][(j+1+N)%N]+lattice[i][(j-1+N)%N])
  return (E_field+E_i)/(2)

def findC(E,T,N): #find heat capacity per spin in equillibrium state
  return np.var(E[100:])/((T*N)**2)

def findTC(C,T):
  return T[list(C).index(np.max(C))]
```

```
#16*16 lattice
T=np.linspace(1.5,3.5,20)
C=np.zeros((5,20))
for i in range(5):
  for j in range(20):
    E1=[]
    LAT1=constrct_lattice(16)
    for x in range(300): #find M after 299 sweeps
      E1.append(findE(LAT1,16,H))
      LAT1=sweep(LAT1,16,T[j],H)
    C[i][j]=findC(E1,T[j],20)
C_f=np.average(C,axis=0)
plt.plot(T,C_f,'o')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Specific heat per spin/k_B')
plt.title('Specific heat per spin vs temperature: lattice 16^2')
print('Critical temperature is found to be',findTC(C_f,T))

"""####32*32 lattice"""

import numpy as np
import random as rd
from math import exp
import matplotlib.pyplot as plt


#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
```

```
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findE(lattice,N,H): #find energy per spin of the lattice
  E_field=-H*np.sum(np.sum(lattice,axis=0))
  E_i=0
  for i in range(N):
    for j in range(N):
      E_i+= -lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+lattice
      ↪  [i][(j+1+N)%N]+lattice[i][(j-1+N)%N])
  return (E_field+E_i)/(2)

def findC(E,T,N): #find heat capacity per spin in equillibrium state
  return np.var(E[100:])/((T*N)**2)

def findTC(C,T):
  return T[list(C).index(np.max(C))]

#32*32 lattice
T=np.linspace(1.5,3.5,20)
C=np.zeros((5,20))
for i in range(5):
  for j in range(20):
    E1=[]
    LAT1=constrct_lattice(32)
    for x in range(300): #find M after 299 sweeps
      E1.append(findE(LAT1,32,H))
      LAT1=sweep(LAT1,32,T[j],H)
    C[i][j]=findC(E1,T[j],20)
C_f=np.average(C,axis=0)
```

```
plt.plot(T,C_f,'o')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Specific heat per spin/k_B')
plt.title('Specific heat per spin vs temperature: lattice 32^2')
print('Critical temperature is found to be',findTC(C_f,T))

"""####64*64 lattice"""

import numpy as np
import random as rd
from math import exp
import matplotlib.pyplot as plt


#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
```

```
  M=sum#/(N**2)
  return M

def findE(lattice,N,H): #find energy per spin of the lattice
  E_field=-H*np.sum(np.sum(lattice,axis=0))
  E_i=0
  for i in range(N):
    for j in range(N):
      E_i+= -lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+lattice
      ↪  [i][(j+1+N)%N]+lattice[i][(j-1+N)%N])
  return (E_field+E_i)/(2)

def findC(E,T,N): #find heat capacity per spin in equillibrium state
  return np.var(E[100:])/((T*N)**2)

def findTC(C,T):
  return T[list(C).index(np.max(C))]


#64*64 lattice
T=np.linspace(1.5,3.5,20)
C=np.zeros((5,20))
for i in range(5):
  for j in range(20):
    E1=[]
    LAT1=constrct_lattice(64)
    for x in range(300): #find M after 299 sweeps
      E1.append(findE(LAT1,64,H))
      LAT1=sweep(LAT1,64,T[j],H)
    C[i][j]=findC(E1,T[j],20)
C_f=np.average(C,axis=0)
plt.plot(T,C_f,'o')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Specific heat per spin/k_B')
plt.title('Specific heat per spin vs temperature: lattice 64^2')
print('Critical temperature is found to be',findTC(C_f,T))

"""### Magnetic susceptibility

####16*16 lattice
"""

import numpy as np
```

```
import random as rd
from math import exp
import matplotlib.pyplot as plt


#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findE(lattice,N,H): #find energy per spin of the lattice
  E_field=-H*np.sum(np.sum(lattice,axis=0))
  E_i=0
  for i in range(N):
    for j in range(N):
```

```
      E_i+= -lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+lattice
      ↪  [i][(j+1+N)%N]+lattice[i][(j-1+N)%N])
  return (E_field+E_i)/(2)

def findC(E,T,N): #find heat capacity per spin in equillibrium state
  return np.var(E[100:])/((T*N)**2)

def findX(M,T,N): #find susceptibility per spin
  return np.var(M[100:])/(T*N*N)
def findTC(C,T):
  return T[list(C).index(np.max(C))]


#16*16 lattice
T=np.linspace(1,4,20)
X=np.zeros((5,20))
for i in range(5):
  for j in range(20):
    M1=[]
    LAT1=constrct_lattice(16)
    for x in range(300): #find M after 299 sweeps
      M1.append(findM(LAT1,16))
      LAT1=sweep(LAT1,16,T[j],H)
    X[i][j]=findX(M1,T[j],16)
X_f=np.average(X,axis=0)
plt.plot(T,X_f,'o')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Susceptibility per spin/k_B')
plt.title('Susceptibility per spin vs temperature: lattice 16^2')
print('Critical temperature is found to be',findTC(X_f,T))

"""####32*32 lattice"""

import numpy as np
import random as rd
from math import exp
import matplotlib.pyplot as plt


#parameter list
H=0

#functions
```

```
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findE(lattice,N,H): #find energy per spin of the lattice
  E_field=-H*np.sum(np.sum(lattice,axis=0))
  E_i=0
  for i in range(N):
    for j in range(N):
      E_i+= -lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+lattice
      ↪  [i][(j+1+N)%N]+lattice[i][(j-1+N)%N])
  return (E_field+E_i)/(2)

def findC(E,T,N): #find heat capacity per spin in equillibrium state
  return np.var(E[100:])/((T*N)**2)

def findX(M,T,N): #find susceptibility per spin
  return np.var(M[100:])/(T*N*N)
def findTC(C,T):
```

```
  return T[list(C).index(np.max(C))]


#32*32 lattice
T=np.linspace(1,4,20)
X=np.zeros((5,20))
for i in range(5):
  for j in range(20):
    M1=[]
    LAT1=constrct_lattice(32)
    for x in range(300): #find M after 299 sweeps
      M1.append(findM(LAT1,32))
      LAT1=sweep(LAT1,32,T[j],H)
    X[i][j]=findX(M1,T[j],32)
X_f=np.average(X,axis=0)
plt.plot(T,X_f,'o')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Susceptibility per spin/k_B')
plt.title('Susceptibility per spin vs temperature: lattice 32^2')
print('Critical temperature is found to be',findTC(X_f,T))

"""####64*64 lattice"""

import numpy as np
import random as rd
from math import exp
import matplotlib.pyplot as plt


#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
```

```
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findE(lattice,N,H): #find energy per spin of the lattice
  E_field=-H*np.sum(np.sum(lattice,axis=0))
  E_i=0
  for i in range(N):
    for j in range(N):
      E_i+= -lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+lattice
      ↪  [i][(j+1+N)%N]+lattice[i][(j-1+N)%N])
  return (E_field+E_i)/(2)

def findC(E,T,N): #find heat capacity per spin in equillibrium state
  return np.var(E[100:])/((T*N)**2)

def findX(M,T,N): #find susceptibility per spin
  return np.var(M[100:])/(T*N*N)
def findTC(C,T):
  return T[list(C).index(np.max(C))]


#64*64 lattice
T=np.linspace(1,4,20)
X=np.zeros((5,20))
for i in range(5):
  for j in range(20):
    M1=[]
    LAT1=constrct_lattice(64)
```

```
    for x in range(300): #find M after 299 sweeps
      M1.append(findM(LAT1,64))
      LAT1=sweep(LAT1,64,T[j],H)
    X[i][j]=findX(M1,T[j],64)
X_f=np.average(X,axis=0)

plt.plot(T,X_f,'o')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Susceptibility per spin/k_B')

plt.title('Susceptibility per spin vs temperature: lattice 64^2')
print('Critical temperature is found to be',findTC(X_f,T))

"""##Task 4: critical exponents

### \alpha
"""

import numpy as np
import random as rd
from math import exp,log
import matplotlib.pyplot as plt
from scipy import stats


#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
```

```
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findE(lattice,N,H): #find energy per spin of the lattice
  E_field=-H*np.sum(np.sum(lattice,axis=0))
  E_i=0
  for i in range(N):
    for j in range(N):
      E_i+= -lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+lattice
      ↪  [i][(j+1+N)%N]+lattice[i][(j-1+N)%N])
  return (E_field+E_i)/(2)

def findC(E,T,N): #find heat capacity in equillibrium state
  return np.var(E[100:])/(T**2)

def findTC(C,T):
  return T[list(C).index(np.max(C))]

#32*32 lattice, T_c found to be 2.3421
T=np.linspace(2.28,2.32,20)
C=np.zeros((5,20))
for i in range(5):
  for j in range(20):
    E1=[]
    LAT1=constrct_lattice(32)
    for x in range(300): #find M after 299 sweeps
      E1.append(findE(LAT1,32,H))
      LAT1=sweep(LAT1,32,T[j],H)
    C[i][j]=findC(E1,T[j],20)
C_f=np.average(C,axis=0)
```

```
plt.plot([(2.3421-t) for t in T], C_f,'o')
plt.xlabel('t/(J/k_B)')
plt.ylabel('Specific Heat')
plt.title('Specific heat vs reduced temperature')

lnC=[log(c) for c in C_f]
lnt=[log(2.3421-t) for t in T]

slope, intercept, r_value, p_value, std_err = stats.linregress(lnt,lnC)
print('Critical exponent \alpha is found to be', -slope, '. Analytic \alpha is
↪  expected to be 0.')

"""### \beta"""

import numpy as np
import random as rd
from math import exp,log
import matplotlib.pyplot as plt
from scipy import stats


#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass
```

```
  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findE(lattice,N,H): #find energy per spin of the lattice
  E_field=-H*np.sum(np.sum(lattice,axis=0))
  E_i=0
  for i in range(N):
    for j in range(N):
      E_i+= -lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+lattice
      ↪  [i][(j+1+N)%N]+lattice[i][(j-1+N)%N])
  return (E_field+E_i)/(2)

def findC(E,T,N): #find heat capacity in equillibrium state
  return np.var(E[100:])/(T**2)

def findTC(C,T):
  return T[list(C).index(np.max(C))]

#32*32 lattice, T_c found to be 2.3421
T=np.linspace(2.28,2.32,20)
M_av=np.zeros((5,20))
for i in range(5):
  for j in range(20):
    M=[]
    LAT1=constrct_lattice(32)
    for x in range(300): #find M after 299 sweeps
      M.append(findM(LAT1,32))
      LAT1=sweep(LAT1,32,T[j],H)
    M_av[i][j]=np.average(M[100:])
M_f=np.average(M_av,axis=0)
lnM=[log(m) for m in M_f]
lnt=[log(2.3421-t) for t in T]

coeffs = np.polyfit(lnt,lnM, 1);
fittedX1 = np.linspace(min(lnt), max(lnt), 200);
fittedY1 = np.polyval(coeffs, fittedX1);
```

```
plt.plot(fittedX1, fittedY1, 'b-');

plt.plot(lnt,lnM,'o')
plt.xlabel('ln(t)')
plt.ylabel('ln(Magnetisation)')
plt.title('ln(Magnetisation) vs ln(t)')

slope, intercept, r_value, p_value, std_err = stats.linregress(lnt,lnM)
print('Critical exponent beta is found to be', slope, '+-',std_err,'. Analytic
↪  beta is expected to be 0.125')

"""### \delta

#### T=5 hysteresis
"""

import numpy as np
import random as rd
from math import exp,log
import matplotlib.pyplot as plt
from scipy import stats

#Above critical temperature
# parameter list

T=5

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
```

```
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum/(N**2)
  return M

Hp=np.linspace(-5,5,30)

Mp=np.zeros((5,30))

for i in range(5):
  for j in range(30):
    LATp=constrct_lattice(32)
    for x in range(150):
      LATp=sweep(LATp,32,T,Hp[j])
    Mp[i][j]=findM(LATp,32)

Mp_f=np.average(Mp,axis=0)

plt.plot(Hp,Mp_f,'o')
plt.xlabel('External field/T')
plt.ylabel('Magnetisation per spin')
plt.title('Magnetisation per spin vs external field at T=5')

plt.show()

"""#### T=1.1 hysteresis"""

import numpy as np
import random as rd
from math import exp,log
import matplotlib.pyplot as plt
from scipy import stats

#below critical temperature
# parameter list
```

```
T=1.1

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum/(N**2)
  return M

Hp=np.linspace(-5,5,30)

Mp=np.zeros((5,30))

for i in range(5):
  for j in range(30):
    LATp=constrct_lattice(32)
    for x in range(150):
      LATp=sweep(LATp,32,T,Hp[j])
    Mp[i][j]=findM(LATp,32)
```

```
Mp_f=np.average(Mp,axis=0)

plt.plot(Hp,Mp_f,'o')
plt.xlabel('External field/T')
plt.ylabel('Magnetisation per spin')
plt.title('Magnetisation per spin vs external field at T=1.1')

plt.show()

"""#### \delta linear regression"""

import numpy as np
import random as rd
from math import exp,log
import matplotlib.pyplot as plt
from scipy import stats


# parameter list

T=2.34

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice
```

```
def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

Hp=np.linspace(0.5,1.5,30)

Mp=np.zeros((5,30))

for i in range(5):
  for j in range(30):
    LATp=constrct_lattice(32)
    for x in range(150):
      LATp=sweep(LATp,32,T,Hp[j])
    Mp[i][j]=findM(LATp,32)

Mp_f=np.average(Mp,axis=0)
lnM=[log(m) for m in Mp_f]
lnH=[log(h) for h in Hp]

plt.plot(lnH,lnM,'o')
plt.xlabel('ln(H)')
plt.ylabel('ln(Magnetisation)')
plt.title('Critical component delta: ln(Magnetisation) vs ln(H)')

coeffs = np.polyfit(lnH,lnM, 1);
fittedX1 = np.linspace(min(lnH), max(lnH), 200);
fittedY1 = np.polyval(coeffs, fittedX1);
plt.plot(fittedX1, fittedY1, 'b-');

slope, intercept, r_value, p_value, std_err = stats.linregress(lnH,lnM)
print('Critical exponent delta is found to be', 1/slope, '. Analytic delta is
↪  expected to be 15')

plt.show()

"""##Task 5: finite size scaling

### Statement 1
"""
```

```
import numpy as np
import random as rd
from math import exp,log
import matplotlib.pyplot as plt
from scipy import stats


#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findE(lattice,N,H): #find energy per spin of the lattice
  E_field=-H*np.sum(np.sum(lattice,axis=0))
  E_i=0
```

```
  for i in range(N):
    for j in range(N):
      E_i+= -lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+lattice
      ↪  [i][(j+1+N)%N]+lattice[i][(j-1+N)%N])
  return (E_field+E_i)/(2)

def findC(E,T,N): #find heat capacity per spin in equillibrium state
  return np.var(E[100:])/((T*N)**2)

def findTC(C,T):
  return T[list(C).index(np.max(C))]


N=[np.int(x) for x in np.linspace(5,65,7)]
T=np.linspace(2.15,2.35,10)
Tc=np.zeros((3,7))
for i in range(3):
  for j in range(7):
    c=[]
    for t in T:
      E1=[]
      LAT1=constrct_lattice(N[j])
      for x in range(200): #find M after 199 sweeps
        E1.append(findE(LAT1,N[j],H))
        LAT1=sweep(LAT1,N[j],t,H)
      c.append(findC(E1,t,N[j]))
    Tc[i][j]=findTC(c,T)

Tc_f=np.average(Tc, axis=0)
N_inv=[1/l for l in N]


slope, intercept, r_value, p_value, intercept_stderr =
↪  stats.linregress(N_inv,Tc_f)

coeffs = np.polyfit(N_inv, Tc_f, 1);
fittedX1 = np.linspace(min(N_inv), max(N_inv), 200);
fittedY1 = np.polyval(coeffs, fittedX1);

plt.plot(N_inv,Tc_f,'o')
plt.plot(fittedX1, fittedY1, 'b-')
plt.xlabel('1/N')
plt.ylabel('Critical Temperature/(J/k_B)')
```

```
plt.title('Critical temperature vs 1/N')

print(r_value,intercept)
print('Tc(inf) is equal to', intercept, '+-', intercept_stderr,'Analytic result
↪  is 2.2692.')
plt.show()

"""###Statement 2"""

import numpy as np
import random as rd
from math import exp,log
import matplotlib.pyplot as plt
from scipy import stats



#parameter list
H=0

#functions
def constrct_lattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N),dtype=int)
  return l

def sweep(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at temp
↪  T with external field strength H
  for i in range(N):
    for j in range(N):
      E_flip=2*lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+latti
      ↪  ce[i][(j+1+N)%N]+lattice[i][(j-1+N)%N]+2*H)
      if E_flip<=0:
        lattice[i][j]=np.int(-lattice[i][j])
      else:
        p=rd.random()
        if p<=exp(-E_flip/T):
          lattice[i][j]=np.int(-lattice[i][j])
        else:
          pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
```

```
  s0=np.sum(lattice,axis=0)
  sum=np.sum(s0)
  M=sum#/(N**2)
  return M

def findE(lattice,N,H): #find energy per spin of the lattice
  E_field=-H*np.sum(np.sum(lattice,axis=0))
  E_i=0
  for i in range(N):
    for j in range(N):
      E_i+= -lattice[i][j]*(lattice[(i+1+N)%N][j]+lattice[(i-1+N)%N][j]+lattice
      ↪  [i][(j+1+N)%N]+lattice[i][(j-1+N)%N])
  return (E_field+E_i)/(2)

def findC(E,T,N): #find heat capacity per spin in equillibrium state
  return np.var(E[100:])/((T*N)**2)

def findTC(C,T):
  return T[list(C).index(np.max(C))]

N=[np.int(x) for x in np.linspace(5,65,7)]
T=np.linspace(2.15,2.35,10)
Cmax=np.zeros((5,7))
for i in range(5):
  for j in range(7):
    c=[]
    for t in T:
      E1=[]
      LAT1=constrct_lattice(N[j])
      for x in range(200): #find M after 199 sweeps
        E1.append(findE(LAT1,N[j],H))
        LAT1=sweep(LAT1,N[j],t,H)
      c.append(findC(E1,t,N[j]))
    Cmax[i][j]=np.max(c)

Cmax_f=np.average(Cmax,axis=0)
lnN2=[log(n) for n in N]

plt.plot(lnN2,Cmax_f,'o')
plt.xlabel('ln(lattice size, N)')
plt.ylabel('Specific heat per spin at T_c')
plt.title('Specific heat per spin at T_c vs ln(lattice size, N)')
slope, intercept, r_value, p_value, std_err = stats.linregress(lnN2,Cmax_f)
```

```
coeffs = np.polyfit(lnN2,Cmax_f, 1);
fittedX1 = np.linspace(min(lnN2), max(lnN2), 200);
fittedY1 = np.polyval(coeffs, fittedX1);

plt.plot(fittedX1, fittedY1, 'b-')

print(r_value)
print('Slope is found to be', slope,'+-',std_err,'and it is expected to be 1.')
plt.show()

"""#Part 3: 3D lattice

##Task 1: decorelation time
"""

import numpy as np
import random as rd
from math import exp,sqrt
import matplotlib.pyplot as plt



#parameter list
H=0

#functions
def constrct_3Dlattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N,N),dtype=int)
  return l



def sweep3D(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at
↪  temp T with external field strength H
  for i in range(N):
    for j in range(N):
      for k in range(N):
        E_flip=2*lattice[i][j][k]*(lattice[(i+1+N)%N][j][k]+lattice[(i-1+N)%N][
        ↪  j][k]+lattice[i][(j+1+N)%N][k]+lattice[i][(j-1+N)%N][k]+lattice[i][
        ↪  j][(k-1+N)%N]+lattice[i][j][(k+1+N)%N]+2*H)
        if E_flip<=0:
          lattice[i][j][k]=np.int(-lattice[i][j][k])
        else:
          p=rd.random()
```

```
          if p<=exp(-E_flip/T):
            lattice[i][j][k]=np.int(-lattice[i][j][k])
          else:
            pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  sum=np.sum(np.sum(np.sum(lattice,axis=0),axis=0))
  M=sum#/(N**2)
  return M

def findAcov(M,tau): #find autocorelation
  M_av=np.mean(M)
  M_prime=[x-M_av for x in M]
  t_tot=len(M)
  A=np.mean([M_prime[x]*M_prime[x+tau] for x in range(t_tot-tau)])
  return A

def findtau_e(M):
  A0=findAcov(M,0)
  for t in range(1,len(M)):
    At=findAcov(M,t)
    a=At/A0
    if a>exp(-1):
      pass
    else:
      return t

#16*16*16 lattice
T=np.linspace(1.5,10,20)
te=np.zeros((5,20))
for i in range(5):
  for j in range(20):
    M=[]
    LAT=constrct_3Dlattice(16)
    for x in range(300):
      LAT=sweep3D(LAT,16,T[j],H)
      M.append(findM(LAT,16))
    te[i][j]=findtau_e(M)
te_f=np.average(te,axis=0)
te_err=np.std(te,axis=0)
```

```
print('Decorreltation time at Tc is found to be',np.max(te_f),'+-',
↪  te_err[list(te_f).index(np.max(te_f))]/sqrt(5))
plt.plot(T,te_f,'o')
plt.title('Decorrelation time r_e vs temperature for lattice 16^3')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Decorrelation time')
plt.show()

"""## Task 2: Critical temperature"""

import numpy as np
import random as rd
from math import exp
import matplotlib.pyplot as plt


#parameter list
H=0

#functions
def constrct_3Dlattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N,N),dtype=int)
  return l


def sweep3D(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at
↪  temp T with external field strength H
  for i in range(N):
    for j in range(N):
      for k in range(N):
        E_flip=2*lattice[i][j][k]*(lattice[(i+1+N)%N][j][k]+lattice[(i-1+N)%N][⌟
        ↪  j][k]+lattice[i][(j+1+N)%N][k]+lattice[i][(j-1+N)%N][k]+lattice[i][⌟
        ↪  j][(k-1+N)%N]+lattice[i][j][(k+1+N)%N]+2*H)
        if E_flip<=0:
          lattice[i][j][k]=np.int(-lattice[i][j][k])
        else:
          p=rd.random()
          if p<=exp(-E_flip/T):
            lattice[i][j][k]=np.int(-lattice[i][j][k])
          else:
            pass
```

```
  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  sum=np.sum(np.sum(np.sum(lattice,axis=0),axis=0))
  M=sum/(N**2)
  return M

# Mean magnetisation is equal to sum of total M divided by the number of steps
T=np.linspace(1.5,8,50)
M_av=[]
for t in T:
  M=[]
  LAT=constrct_3Dlattice(16)
  for x in range(200): #find M after 199 sweeps
    M.append(findM(LAT,16))
    LAT=sweep3D(LAT,16,t,H)
  M_av.append(np.mean(M[100:]))



def CT(M_f,T):
  p=np.isclose(M_f,np.zeros(50),atol=0.5)
  id=0
  for i in p:
    if i == False:
      pass
    else:
      id=list(p).index(i)
      break
  return T[id]
print('Critical temperature is found to be',CT(M_av,T))

plt.plot(T,M_av,'o')
plt.title('Magnetisation per spin vs Temperature: lattice 16^3')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Magnetisation per spin')

"""## Task 3: Specific heat"""

import numpy as np
import random as rd
from math import exp
import matplotlib.pyplot as plt
```

```
#parameter list
H=0

#function
def constrct_3Dlattice(N): #set up an N*N lattice with all spins alligned upward
  l=np.ones((N,N,N),dtype=int)
  return l


def sweep3D(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at
↪  temp T with external field strength H
  for i in range(N):
    for j in range(N):
      for k in range(N):
        E_flip=2*lattice[i][j][k]*(lattice[(i+1+N)%N][j][k]+lattice[(i-1+N)%N][
        ↪  j][k]+lattice[i][(j+1+N)%N][k]+lattice[i][(j-1+N)%N][k]+lattice[i][
        ↪  j][(k-1+N)%N]+lattice[i][j][(k+1+N)%N]+2*H)
        if E_flip<=0:
          lattice[i][j][k]=np.int(-lattice[i][j][k])
        else:
          p=rd.random()
          if p<=exp(-E_flip/T):
            lattice[i][j][k]=np.int(-lattice[i][j][k])
          else:
            pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  sum=np.sum(np.sum(np.sum(lattice,axis=0),axis=0))
  M=sum#/(N**2)
  return M

def findE(lattice,N,H): #find energy per spin of the lattice
  E_field=-H*np.sum(np.sum(lattice,axis=0))
  E_i=0
  for i in range(N):
    for j in range(N):
      for k in range(N):
```

```
        E_i+= -lattice[i][j][k]*(lattice[(i+1+N)%N][j][k]+lattice[(i-1+N)%N][j]
        ↪  [k]+lattice[i][(j+1+N)%N][k]+lattice[i][(j-1+N)%N][k]+lattice[i][j]
        ↪  [(k-1+N)%N]+lattice[i][j][(k+1+N)%N])
  return (E_field+E_i)/(2)

def findC(E,T,N): #find heat capacity per spin in equillibrium state
  return np.var(E[100:])/((T*N)**2)

def findTC(C,T):
  return T[list(C).index(np.max(C))]


#16*16*16 lattice
T=np.linspace(3,6,30)
C=np.zeros((5,30))
for i in range(5):
  for j in range(30):
    E1=[]
    LAT1=constrct_3Dlattice(16)
    for x in range(200): #find M after 299 sweeps
      E1.append(findE(LAT1,16,H))
      LAT1=sweep3D(LAT1,16,T[j],H)
    C[i][j]=findC(E1,T[j],16)
C_f=np.average(C,axis=0)
plt.plot(T,C_f,'o')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Specific heat per spin/k_B')
plt.title('Specific heat per spin vs temperature: lattice 16^3')
print(findTC(C_f,T))

"""##Task 4: Susceptibility"""

import numpy as np
import random as rd
from math import exp
import matplotlib.pyplot as plt


#parameter list
H=0

#function
def constrct_3Dlattice(N): #set up an N*N lattice with all spins alligned upward
```

```
  l=np.ones((N,N,N),dtype=int)
  return l


def sweep3D(lattice,N,T,H): #perform 1 complete sweep over a N*N lattice at
↪  temp T with external field strength H
  for i in range(N):
    for j in range(N):
      for k in range(N):
        E_flip=2*lattice[i][j][k]*(lattice[(i+1+N)%N][j][k]+lattice[(i-1+N)%N][
        ↪  j][k]+lattice[i][(j+1+N)%N][k]+lattice[i][(j-1+N)%N][k]+lattice[i][
        ↪  j][(k-1+N)%N]+lattice[i][j][(k+1+N)%N]+2*H)
        if E_flip<=0:
          lattice[i][j][k]=np.int(-lattice[i][j][k])
        else:
          p=rd.random()
          if p<=exp(-E_flip/T):
            lattice[i][j][k]=np.int(-lattice[i][j][k])
          else:
            pass

  return lattice

def findM(lattice,N): #use this function to find total normalized magnetisation
↪  such that when fully alligned M=+-1
  sum=np.sum(np.sum(np.sum(lattice,axis=0),axis=0))
  M=sum#/(N**2)
  return M

def findE(lattice,N,H): #find energy per spin of the lattice
  E_field=-H*np.sum(np.sum(lattice,axis=0))
  E_i=0
  for i in range(N):
    for j in range(N):
      for k in range(N):
        E_i+= -lattice[i][j][k]*(lattice[(i+1+N)%N][j][k]+lattice[(i-1+N)%N][j]
        ↪  [k]+lattice[i][(j+1+N)%N][k]+lattice[i][(j-1+N)%N][k]+lattice[i][j]
        ↪  [(k-1+N)%N]+lattice[i][j][(k+1+N)%N])
  return (E_field+E_i)/(2)

def findC(E,T,N): #find heat capacity per spin in equillibrium state
  return np.var(E[100:])/((T*N)**2)
```

```
def findX(M,T,N): #find heat capacity per spin in equillibrium state
  return np.var(M[100:])/((T*N*N)

def findTC(C,T):
  return T[list(C).index(np.max(C))]

#16*16*16 lattice
T=np.linspace(3,6,40)
X=np.zeros((5,40))
for i in range(5):
  for j in range(40):
    M1=[]
    LAT1=constrct_3Dlattice(16)
    for x in range(300): #find M after 299 sweeps
      M1.append(findM(LAT1,16))
      LAT1=sweep3D(LAT1,16,T[j],H)
    X[i][j]=findX(M1,T[j],16)
X_f=np.average(X,axis=0)
plt.plot(T,X_f,'o')
plt.xlabel('Temperature/(J/k_B)')
plt.ylabel('Susceptibility per spin/k_B')
plt.title('Susceptibility per spin vs temperature: lattice 16^3')
print(findTC(X_f,T))
```