Obtaining Critical Exponents of the Ising Model via Cluster Monte Carlo Analysis

Jyotirmai (Joe) Singh - Physics 212 Final Project

Abstract

In this paper we consider the behaviour of the Ising model near the phase transition for $d = 2-5$. Using cluster Monte Carlo techniques (in particular the Wolff algorithm), we simulate the magnetisation, susceptibility, and heat capacity of the square Ising model. Then, by using finite size scaling analysis, we are able to determine the critical exponents β, γ, ν, and α as well as the critical temperature T_c. Our values are generally in good agreement with published results for $d = 2, 3$ although we see some tension in the heat capacity. For $d = 4, 5$ we compare our results with the predictions of Mean Field Theory. We find that while the $d = 4$ case demonstrates mean field behaviour, in the 5D case the MFT predictions were not as readily verified, largely due to the complexity issues associated with simulating large statistics at high dimensions.

Introduction

One of the central concepts in the study of statistical mechanics is the analysis of phase transitions in varied and exotic systems. For this reason, the Ising Model has attracted immense attention ever since its introduction by Wilhelm Lenz and Ernst Ising. The model consists of a set of lattice sites forming a d-dimensional hypercubic lattice. At every lattice site i we place a discrete variable $s_i \in \{+1, -1\}$. The Hamiltonian of an Ising lattice is given by

$$\mathcal{H} = -J \sum_{\langle ij \rangle} s_i s_j + h \sum_i s_i \tag{1}$$

where J is a coupling constant and h is an external magnetic field. We shall work in the case where $h = 0$ throughout this paper as this aids in allowing one to examine the critical exponents at the phase transitions without the complication of an external symmetry breaking field. In the interaction term, the notation $\langle ij \rangle$ denotes a sum over the nearest neighbours s_j of the point s_i. Intuitively, it is easy to see that in the low temperature case the preferred configurations is to have the spins all aligned and that at high temperatures a randomised spin configuration is optimal. The former is known as the magnetised phase and the latter the demagnetised phase, and the

transition between these is the quintessential example of a phase transition. In a somewhat more formal sense, the phase transition occurs when the partition function $Z = \exp(-\beta\mathcal{H})$ does not behave analytically, leading to discontinuities in the free energy $F \propto \ln(Z)$.

Critical Exponents

Having established the importance of phase transitions, it is necessary for us to now introduce the tools through which to analyse them. In particular, the non analytic behaviour of various thermodynamic functions is captured by a set of *critical exponents* [1]. Below we describe the exponents which we will attempt to obtain via Monte Carlo methods. Define the reduced temperature $t = \frac{T-T_C}{T}$ which parametrises how close one is to the critical point T_C.

1. The magnetisation exponent β
   Define the magnetisation m as
   $$m = \frac{1}{N}\left|\sum_i s_i\right| \tag{2}$$
   where N is the number of sites in the lattice. If all the spins are aligned, $m = 1$ and if they are uncorrelated, then $m = 0$. In this sense it measures the amount of disorder in the system, with the former limit characterising the ordered magnetised phase and the latter the disordered demagnetised phase. Near the phase transition, the magnetisation behaves as $m \sim |t|^{\beta}$.

2. The susceptibility exponent γ
   Define the magnetic susceptibility as $\chi = \frac{\partial m}{\partial h}$. It can be shown via manipulation of the partition function that $\chi = \beta\left(\langle M^2\rangle - \langle M\rangle^2\right)$. Near the critical point, the susceptibility goes as $\chi \sim |t|^{-\gamma}$.

3. The correlation length exponent ν
   Let $\langle s_0 s_r\rangle$ denote the correlation function between two random spins. Then the correlation length ξ is defined via the relation $\langle s_0 s_r\rangle = e^{-\frac{r}{\xi}}$. Along with the order parameter (in our case the magnetisation), the correlation length serves as an indicator of a phase transition as it diverges near the critical temperature. In particular, near the transition the correlation length is governed by $\xi \sim |t|^{-\nu}$.

4. The heat capacity exponent α
   Let E be the energy of the Ising system, given of course by the Hamiltonian (1). The heat capacity C is then given by [2]
   $$C = \beta^2\left(\langle E^2\rangle - \langle E\rangle^2\right) \tag{3}$$
   The heat capacity also diverges near the critical point and this divergence is governed by the relation $C \sim |t|^{-\alpha}$.

Mean Field Theory

One particularly powerful method to obtain estimates of critical exponents for models such as the Ising model is the application of Mean Field Theory (MFT). MFT attempts to solve such problems by studying instead a simplified version of the model by averaging over degrees of freedom. For instance, in the Ising model, the effect of all spins on a particular spin is treated as a single averaged effect. This is very useful as it reduces an often intractable many body problem to a feasible one body problem albeit at the cost of some accuracy.

To see how this manifests mathematically, consider the Ising Hamiltonian defined in (1). Let $m = \langle s_i \rangle$ denote the mean value of our spin variables, and let $\delta s_i = s_i - m_i$ denote the fluctuation from the mean. We can then rewrite the Hamiltonian as

$$\mathcal{H} = -J \sum_{\langle ij \rangle} (m_i + \delta s_i)(m_j + \delta s_j) - h \sum_i s_i \tag{4}$$

Expanding this, and neglecting second order terms in the fluctuation of the form $\delta s_i \delta s_j$ (the mean field approximation) yields

$$\mathcal{H} \approx -J \sum_{\langle ij \rangle} (m_i m_j + m_i \delta s_j + m_j \delta s_i) - h \sum_i s_i$$

One of the underlying symmetries of the Ising Model in the thermodynamic limit is that it should be translationally invariant. This means that the average spin should be the same at all sites. Therefore we set $m_i = m_j = m$ and obtain the mean field Hamiltonian

$$\mathcal{H}^{MF} = -J \sum_{\langle ij \rangle} (m^2 + 2m(s_i - m)) - h \sum_i s_i \tag{5}$$

For a d dimensional cubic lattice let $z = 2d$ be the number of neighbours that each spin has. Then simplifying the sum over nearest neighbours and correcting for double counting yields the final mean field Hamiltonian.

$$\mathcal{H}^{MF} = \frac{Jm^2 Nz}{2} - (h + mJz) \sum_i s_i \tag{6}$$

Observe that this is effectively a sum of one body Hamiltonians – the mean field treatment has decoupled the original multi body Hamiltonian and introduced a new effective field $h_{\text{eff}} = h + mJz$.

Using this new uncoupled Hamiltonian, we can get partition function as

$$Z = e^{\frac{-\beta J m^2 nZ}{2}} \left[2 \cosh\left(\beta(h + mJz)\right)\right]^N \tag{7}$$

And finally, we can get the free energy per site

$$f = -\frac{1}{\beta N} \ln(Z) = \frac{1}{2} Jm^2 z - \frac{1}{\beta} \ln\left(2 \cosh\left(h + Jmz\right)\right) \tag{8}$$

Using this, we can begin to analyse the behaviour of the critical exponents. To illustrate this, let us focus on the magnetisation m. The magnetisation can be obtained from the free energy per site as $m = -\left(\frac{\partial f}{\partial h}\right) = \tanh(\beta(mJz + h))$. In the $h = 0$ limit that we consider throughout this paper, this reduces to $m = \tanh(\beta m J z)$. As we change the temperature/dimension of the lattice, the value of the factor $\beta J z$ will vary and lead to different possible values for the magnetisation as illustrated in Figure 1.

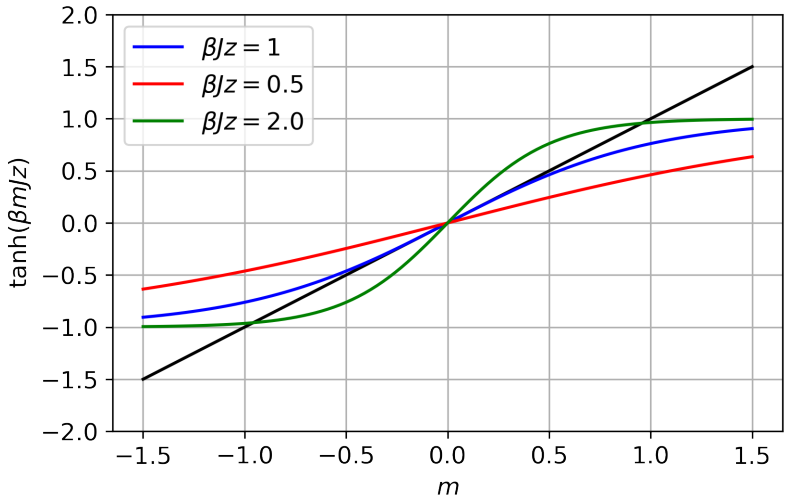


Figure 1: The mean field solution for the magnetisation as a function of varying $\beta J z$

.

Observe that for $\beta J z \leq 1$, the only possible solution is the one with $m = 0$, i.e. the demagnetised phase while for $\beta J z > 1$ a new magnetised phase becomes available. The critical point of transition lies at $\beta J z = 1$ and thus we see a concrete example of a mean field treatment of phase transitions.

To get the critical exponent β, begin by expanding the free energy f around $m = 0$. Since $h = 0$, the expansion assumes the form

$$f \approx c + \frac{t}{2}m^2 + um^4 + \dots \tag{9}$$

Where c and u are (possibly temperature dependant) coefficients and t is the reduced temperature from earlier. Minimising with respect to m and solving for the minimum yields

$$m = \left(\frac{-t}{4u}\right)^{\frac{1}{2}}$$

From this we can draw two immediate conclusions. Firstly, the critical exponent can only exist for $t < 0$, i.e. below the critical temperture. This reflects the fact that above the transition the magnetisation is simply 0 and does not vary at all. Secondly, $m \sim |t|^{\frac{1}{2}}$, meaning that the MFT prediction is $\beta = \frac{1}{2}$.

MFT Predictions for Critical Exponents

A similar analysis can be done for the susceptibility, correlation length, and heat capacity leading to predictions for the exponents $\beta, \nu, \gamma, \alpha$, which are presented in Table 1 [1].

Exponent	MFT Prediction
β	0.5
ν	0.5
γ	1
α	0

Table 1: The critical exponents as predicted by MFT.

The accuracy of the above MFT exponents is dependant on the dimensionality of the Ising model. In particular, the MFT exponents are expected to be correct for $d \geq 4$ and inaccurate for lower dimensions. This is related to the fact that above 4 dimensions the saddle point approximation to obtain the partition function is valid and so MFT should provide accurate predictions [1]. A renormalisation group analysis also yields the same conclusion, with the u coupling mapping to $u' = b^{4-d}u$ where d is the dimensionality and b is the renormalising scale. If $d \geq 4$ this becomes an irrelevant perturbation and MFT continues to hold but in the lower dimensional case this is a relevant perturbation and limits the validity of an MFT approximation.

Monte Carlo Methods

Having established the theoretical basis behind critical exponents and their predicted values in a mean field framework, we now turn to the main task of computing these critical exponents via simulation. The standard paradigm for such simulations is the Markov Chain Monte Carlo (MCMC) method. Each state in the Markov Chain represents a possible state of the system and the Monte Carlo simulation evolves this across iterations based on the transition probabilities between states. A very simple example of a Markov Chain is shown in Figure 2 [3].

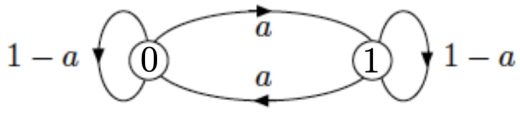


Figure 2: A simple two state symmetric Markov chain.

A MC simulation of the above system would consist of starting randomly in either state 0 or 1 and switching states depending on the transition probability. For a sufficiently large number of iterations, the simulation should end in either state exactly half the time since the Markov chain is symmetric in both of its states. Typically the MCMC algorithm of choice for simulations in statistical mechanics is the Metropolis-Hastings algorithm [4] where starting from an initial lattice state, a randomly chosen spin is flipped. This change is automatically accepted if this spin flip results in a reduction of the energy. If it results in an increase, the acceptance is conditional based on the Boltzmann factor $e^{-\beta(E_f - E_i)}$ where E_f is the final energy and E_i is the initial energy.

The Metropolis-Hastings algorithm outlined as above will correctly simulate the lattice, but faces the significant issue of "critical slowing down" near the critical point [5]. Near the critical point, in a truly infinite system the correlation length diverges to infinity. However, any computer simulation must be finite and a finite system cannot demonstrate a true phase transition since the correlation length is $O(L)$ where L is the linear dimension of the lattice. For large enough L however, the statistical approximation becomes better but then it becomes more difficult to generate statistically independent configurations as the size of ordered regions of the lattice near T_C becomes large so the time τ needed for a region to lose coherence increases as well. For L sufficiently large enough, this time increases as L^z where z is the dynamical critical exponent. For conventional MC methods, $z \sim 2$ [6] so at larger system sizes there is a significant time penalty in the region of interest near the critical point.

Cluster Monte Carlo

As opposed to flipping individual spins, cluster algorithms flip large clusters of spins simultaneously. In this way they bypass the issue of dynamical slowing and allow simulation of larger lattices without an intensive time penalty. The two most prominent cluster algorithms are the Swendsen-Wang and Wolff algorithms [7, 8]. The Wolff algorithm is an improvement over Swendsen-Wang as it has a larger probability of flipping larger clusters. Due to this, we use the Wolff algorithm in our Monte Carlo simulation. The algorithm is as follows:

1. Choose a random starting site i and add it to the cluster.
2. For all of the neighbours of i, check if they have the same spin as i. If yes, add them to the cluster with probability $p = 1 - e^{-2\beta J}$.
3. Repeat step 2 for all spins added to the cluster, but do not perform a check to add them to the cluster if they have been checked already.
4. Repeat until thermalisation of the system.

In the above algorithm, thermalisation effectively means that the system has been allowed to evolve enough such that variables of interest such as the magnetisation assume values from their

steady state distribution. Using our simple Markov chain in Figure 2 as an example, thermalisation here would entail simulating the system for enough iterations such that the probability of the simulation finishing in state 1 or state 0 is exactly $\frac{1}{2}$ since that is the steady state probablity distribution of the Markov chain.

Finite Size Scaling

As mentioned above, a finite lattice cannot display a true phase transition as the correlation length is always constrained by the length scale L of the lattice. Nevertheless, the technique of finite size scaling (FSS) allows finite systems to be used to extrapolate to the infinite system.

Suppose our lattice undergoes a critical transition at $T = T_c$. As noted, in a truly infinite lattice this is characterised by ξ diverging. Suppose the quantity of interest A scales as $A \sim |t|^{-\zeta}$ where ζ is a critical exponent. The infinite lattice critical behaviour should hold to a good approximation even in the finite system, provided that the correlation length $\xi << L$. Since the correlation length goes as $\xi \sim |t|^{-\nu}$ near the critical point, it must be that $A \sim \xi^{\frac{\zeta}{\nu}}$. However, when we approach the point where $L << \xi$, L begins to cutoff the behaviour of A, so the behaviour changes such that $A \sim L^{\frac{\zeta}{\nu}}$. This behaviour is captured in the finite-size scaling ansatz [9]

$$A = \xi^{\frac{\zeta}{\nu}} f\left(\frac{L}{\xi}\right)$$

where the scaling function $f(x)$ is governed by the following behaviour:

$$f(x) = \begin{cases} \text{const.} & \text{if } |x| >> 1 \\ \sim x^{\frac{\zeta}{\nu}} & \text{if } x \to 0 \end{cases}$$

However observe that the scaling function still depends on the infinite system correlation length ξ which we do not know *a priori*. To offset this, we introduce a modified scaling function $\tilde{f}(x) = x^{-\zeta} f\left(x^{\nu}\right)$. With this new function, the scaling ansatz now assumes the form

$$A = L^{\frac{\zeta}{\nu}} \tilde{f}\left(L^{\frac{1}{\nu}} t\right)$$

with

$$\tilde{f}(x) = \begin{cases} \text{const.} & \text{if } x \to 0 \\ \sim L^{-\frac{\zeta}{\nu}} t^{-\zeta} & \text{if } |x| >> 1 \end{cases}$$

Data Collapse

A simulation yields $A(L, t)$, i.e. the quantity of interest on a lattice of size L and at temperature t. The scaling function is given by

$$\tilde{f}\left(L^{\frac{1}{\nu}}t\right) = L^{-\frac{\zeta}{\nu}}A(L,t)$$

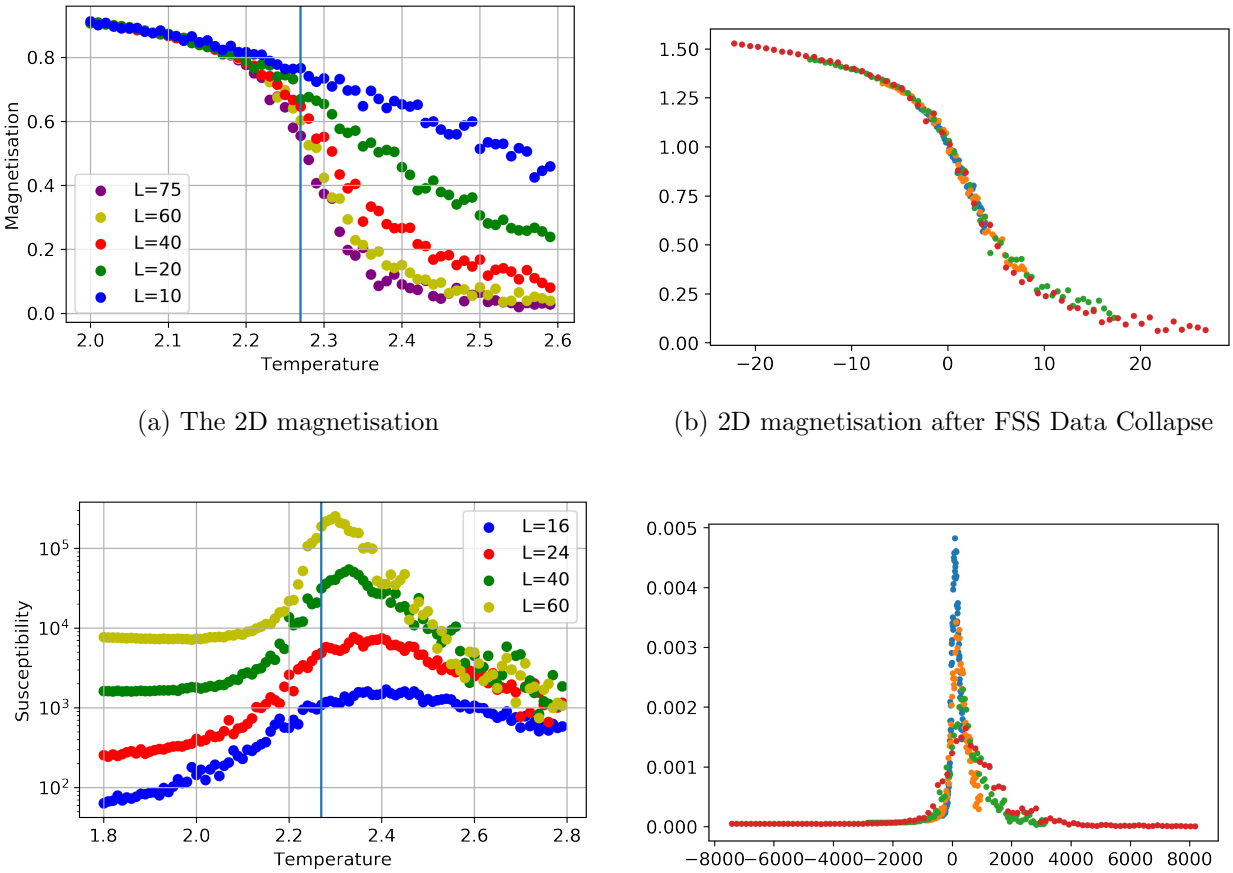
The key to the FSS method is that plotting the right hand side of the above equation vs $L^{\frac{1}{\nu}}t$ should yield the scale invariant curve $\tilde{f}(x)$ always. For there to be true scale invariance however, the critical temperature T_c (which recall comes into the definition of $t = T - T_c$) and the critical exponents ζ and ν must be correct. The scaling functions for the quantities of interest m, χ, and C are given by

$$\tilde{\chi} = L^{-\frac{\gamma}{\nu}}\chi(L,t)$$
$$\tilde{m} = L^{\frac{\beta}{\nu}}m(L,t)$$
$$\tilde{C} = L^{-\frac{\alpha}{\nu}}C(L,t)$$

Practically, this reduces to sampling from the parameter space spanned by T_c, ζ, and ν and essentially minimising the difference between the scaling functions for the various lattice sizes. Since FSS is such a crucial element of our analysis, it is vital that we are able to scan the parameter space spanned by T_c, ζ, and ν very finely and accurately determine the point at which the collapse occurs. After a few attempts at manually implementing a routine to carry out FSS, it became clear that a more sophisticated package was needed to carry this out in order to obtain critical exponents. Therefore, it was decided to use the pyfssa package [10, 11] to aid in the FSS analysis. An example of the finite size analysis with pyfssa integrated into our analysis pipeline is shown in Figure 3 for the magnetisation and susceptibility of the 2D Ising model.

Implementation

The simulation was implemented primarily in C++ while the analysis was done in Python. Initially Python was used for both phases but it became necessary to switch to C++ for higher dimensions/statistics as performance and speed became more essential. The basis for our lattice was the C++ `std::vector` class, and our implementation enforced periodic boundary conditions on the lattice. A further optimisation of our simulation code is that simulations for different temperatures are carried out in parallel using the thread control ability that C++ provides. In each simulation we define a range of temperatures near the critical temperature which is large enough to establish a trend but not too far from the critical point so that finite size scaling analysis is possible on the data. The value of J is always set to 1 since we are primarily concerned about the impact of temperature only. For a given temperature, we initialise a random lattice of the required dimension. We then run the Wolff algorithm on the lattice for 1000 iterations. At each iteration, we store the value of quantities of interest, such as the magnetisation or the energy (for the heat



(a) The 2D magnetisation

(b) 2D magnetisation after FSS Data Collapse

(c) The 2D susceptibility

(d) 2D susceptibility after FSS Data Collapse

Figure 3: The process of data collapse as illustrated by FSS analysis with the 2D magnetisation and susceptibility

capacity), and then we average the stored values in order to compute the average value. Note that we do not store the calculated values for the first 100 iterations as the lattice must first be allowed to thermalise as discussed above. Thus 100 iterations are used to allow the lattice to thermalise and the remaining 900 iterations are averaged to obtain the variables of interest, which are the M, M^2, E, and E^2. This outputs the quantities of interest as functions of the input temperatures. This analysis is repeated for different length sizes and different dimensions. One key limitation is that since increasing the dimension exponentially increases the number of lattice points, at higher dimensions due to the limited computing resources available it was necessary to use lower lattice sizes.

The simulation output data is then transferred to a jupyter notebook where we undertake finite size scaling analysis to obtain the critical exponents, along with errors from the process of gauging where the collapse is optimal, done using the Nelder-Mead algorithm [12].

Results

2D

Value	Result	Accepted Value [13]	Lattice Sizes
β	0.112 ± 0.04	0.125	10, 20, 40, 60, 75
ν	0.96 ± 0.2	1.0	10, 20, 40, 60, 75
γ	1.95 ± 0.3	1.75	16, 24, 40, 60
α	0.0002 ± 0.00005	0.0	10, 20, 50, 60
T_c	2.27 ± 0.002	2.269	10, 20, 40, 60, 75

Table 2: The 2D critical exponents as obtained from our simulations and the values accepted in the literature. Lattice size used refers to the linear dimension L. The number of points N is given by L^2.

Table 2 illustrates the results for the 2D case. Here, we see good agreement between our results and those of the literature. The first three exponents within error of the accepted values. The value for α, while very close to 0, is still in a 4σ tension with it. This suggests that perhaps higher statistics might have been necessary to get a better result. Figure 4 illustrates our data for the 2D heat capacity. As expected, with increasing lattice size the behaviour becomes more divergent around the accepted critical point $T = 2.27$ and slowly converges towards it. For instance the peak for $L = 60$ is notably closer to the critical point than that at $L = 40$ and is more distinctly peaked. This suggests that adding a lattice of larger size to this set may have improved the value of α.

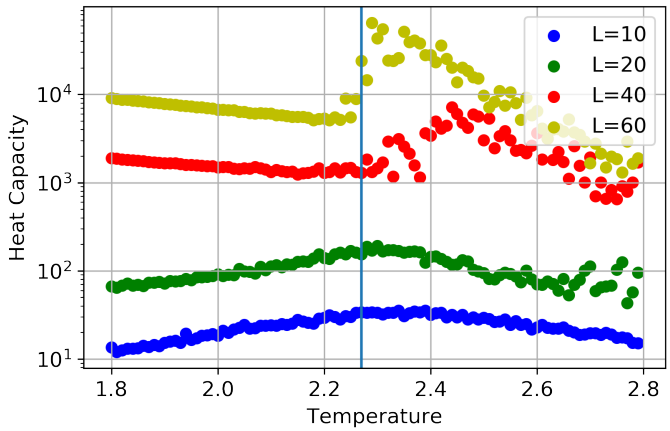


Figure 4: The 2D heat capacity as a function of temperature for various lattice sizes.The vertical line indicates the location of the phase transition.

3D

Value	Result	Accepted Value [14]	Lattice Sizes
β	0.33 ± 0.07	0.326	10, 15, 20
ν	0.62 ± 0.09	0.63	10, 15, 20
γ	1.24 ± 0.1	1.239	8, 16, 20
α	0.116 ± 0.001	0.110	10, 15, 20
T_c	4.50 ± 0.009	4.511	10, 15, 20

Table 3: The 3D critical exponents as obtained from our simulations and the values accepted in the literature. Lattice size used refers to the linear dimension L.

In the three dimensional case our values seem to demonstrate good agreement overall again as shown in table 3. There is a significant tension however for the heat capacity exponent α again, this time at the 6σ level. This seems to reinforce the conclusion we drew in the 2 dimensional case that perhaps our heat capacity simulation was more sensitive to the number of statistics used and that adding a larger lattice to the finite size scaling could have led to a better final value.

4D & 5D Comparison to Mean Field Theory

Table 4 presents the results for the 4D and 5D lattice together and also provides a comparison with the predicted values from MFT.

Value	4D Result	5D Result	MFT Value	4D Lattice Sizes	5D Lattice Sizes
β	0.51 ± 0.03	0.28 ± 0.05	0.5	5, 8, 10	5, 7
ν	0.47 ± 0.03	0.53 ± 0.05	0.5	5, 8, 10	5, 7
γ	1.14 ± 0.04	1.31 ± 0.03	1.00	5, 8, 10	5, 8
α	0.0002 ± 0.00003	0.0004 ± 0.0005	0.0	5, 8	5, 8
T_c	6.69 ± 0.007	8.56 ± 0.05	6.68/8.77	5, 8, 10	5, 7

Table 4: The 4D and 5D critical exponents as obtained from our simulations and the values accepted in the literature. Lattice size used refers to the linear dimension L.

In general, the 4D parameters seem to be in agreement with the predicted values. γ and α are exceptions to this however, demonstrating discrepancies of 3.5σ and $\sim 7\sigma$ respectively. The discrepancy in the susceptibility exponent can probably be attributed to statistical limitations, however the α discrepancy is more serious. Indeed, given the trend of the α exponent displaying

significant tension suggests there may be some minor but non negligible systematic error that we have discounted. This particular discrepancy for 4D could also have been compouned due to the fact that only two lattices were used in the analysis.

In the 5D case the agreement is more ambivalent. The value for ν is in statistical agreement with prediction but apart from that the remaining values are only roughly near the expected values. This is almost certainly linked to the fact that it was difficult to simulate multiple lattices in 5D due to the exponentially higher memory/time cost. Note that for all of these simulations we could only simulate two lattices of a low lattice size. Data at higher lattice sizes would certainly have helped the FSS analysis to converge to a more correct value as it did for the lower dimensions.

Conclusion

In this paper, we have employed Monte Carlo techniques to simulate the behaviour of the Ising model near the critical temperature for a range dimensions. The use of cluster Monte Carlo techniques as opposed to regular Markov Chain MC methods such as the Metropolis-Hastings algorithm allowed us to access higher statistics by avoiding the problem of critical slowdown in the region of interest near the phase transition. More practically, we also attempted to enhance our capacity for higher statistics by parallelising our code so that we could execute our simulations more quickly. Using these Monte Carlo approaches together with finite size scaling techniques allowed us to extract the critical exponents β, ν, γ, and the critical temperature T_c to a good degree of accuracy, although there was some consistent tension with the values of α. In $d = 2, 3$, our extracted exponents are in agreement with the values in the literature and we were also able to verify the predictions of Mean Field Theory in $d = 4$. Our results for $d = 5$ however were less accurate when compared with the MFT prediction. This can be attributed to the difficulty of using larger lattice sizes and thus giving the FSS algorithm sufficient input to converge to the right values.

There exist a number of improvements that could benefit future work of this nature. The most naive improvement is to simply rerun everything with higher statistics, including bigger lattices and more Monte Carlo iterations. In our work we used 1000 iterations with 100 thermalisation iterations which appears to be sufficient as indicated by the good agreement in critical temperatures and exponents, but nevertheless we could have implemented some functionality to compute the autocorrelation between the variables of interest during the MC iterations and used this number to inform when to stop our simulations. Had this analysis revealed fewer iterations were required, we could have more easily simulated larger size lattices. Another improvement could be to run the simulations on a computing cluster with more cores. On a conceptual level, it would be very interesting to run similar simulations on lattices of different shape, such as hexagonal and triangular lattices, and test the universality hypothesis by comparing critical exponents.

Despite fundamental obstacles such as limited computing power, we were nevertheless able to

obtain a good degree with the published values of the critical exponents for $d = 2, 3$. Moreover, we were able to confirm mean field behaviour in $d = 4$ and see some traces of it in $d = 5$, demonstrating the incredible utility of Mean Field Theory in higher dimensions where it allows accurate calculation of critical exponents at a fraction of the computational cost required for other methods such as brute simulation.

References

[1] M. Kardar *Statistical Physics of Fields.* Cambridge University Press, 2007.

[2] P.H. Lundow, K. Markström *The discontinutity of the specific heat for the 5D Ising Model*, Nuclear Physics B 895 (2015) 305-318. DOI: 10.1016/j.nuclphysb.2015.04.013

[3] A. Sinclair, Y. Song *Discrete Mathematics and Probability Theory Note 20: Finite Markov Chains* URL: `https://www.eecs70.org/static/notes/n20.pdf`

[4] W. K. Hastings *Monte Carlo sampling methods using Markov chains and their applications*, Biometrika, Volume 57, Issue 1, April 1970, Pages 97–109. DOI: 10.1093/biomet/57.1.97

[5] H. Gould, J. Tobochnik *Overcoming Critical Slowing Down*, Computers in Physics 3, 82 (1989). DOI: 10.1063/1.4822858

[6] J. K. Williams *Monte Carlo estimate of the dynamical critical exponent of the 2D kinetic Ising model*, J K Williams 1985 J. Phys. A: Math. Gen. 18 49

[7] R. H. Swendsen, and J.-S. Wang, *Nonuniversal critical dynamics in Monte Carlo simulations*, Phys. Rev. Lett., 58(2):86–88 (1987). DOI: 10.1103/PhysRevLett.58.86

[8] U. Wolff, *Collective Monte Carlo Updating for Spin Systems*, Physical Review Letters, 62 (4): 361–364 (1989). DOI: 10.1103/PhysRevLett.62.361

[9] M. E. J. Newman and G. T. Barkema. *Monte Carlo Methods in Statistical Physics. Oxford University Press*, 1999.

[10] Sorge, A . pyfssa 0.7.6. Zenodo (2015). DOI: 10.5281/zenodo.35293

[11] Melcher, O. *autoScale.py - A program for automatic finite-size scaling analyses: A user's guide* (2009), arXiv:0910.5403

[12] Nelder, J. A. and Mead R. *A Simplex Method for Function Minimization* The Computer Journal, Volume 7, Issue 4, January 1965, Pages 308–313. DOI: 10.1093/comjnl/7.4.308

[13] Baxter, R. J. *Exactly Solved Models in Statistical Mechanics*, Academic Press (1982)

[14] Campostri, M. Pelissetto, A. Rossi, P. Vicari, E. *25th-order high-temperature expansion results for three-dimensional Ising-like systems on the simple-cubic lattice* Phys. Rev. E 65, 066127 (2002). DOI: 10.1103/PhysRevE.65.066127

Appendix A - Supplementary Plots

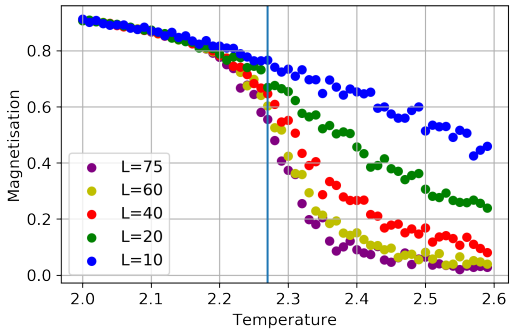


Figure 5: 2D Magnetisation

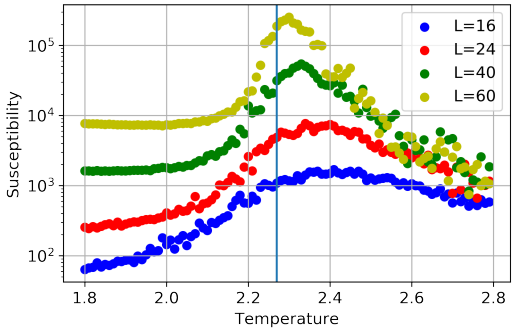


Figure 6: 2D Susceptibility

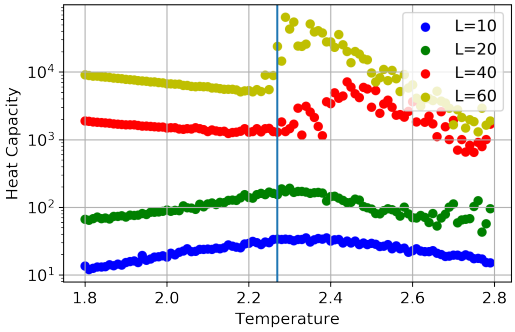


Figure 7: 2D Heat Capacity

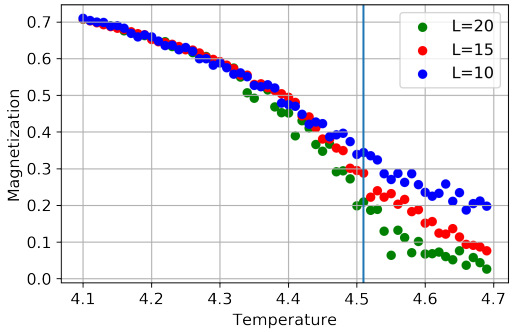


Figure 8: 3D Magnetisation

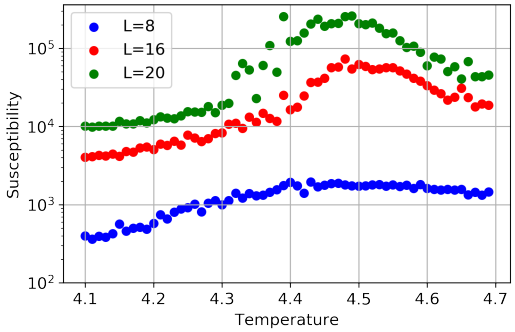


Figure 9: 3D Susceptibility

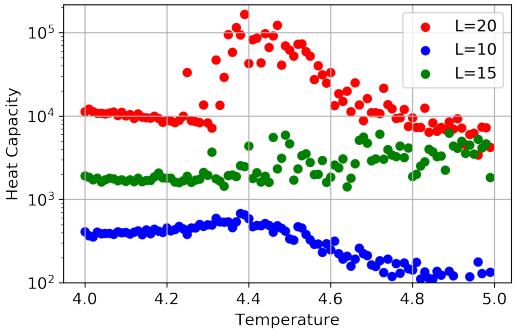


Figure 10: 3D Heat Capacity

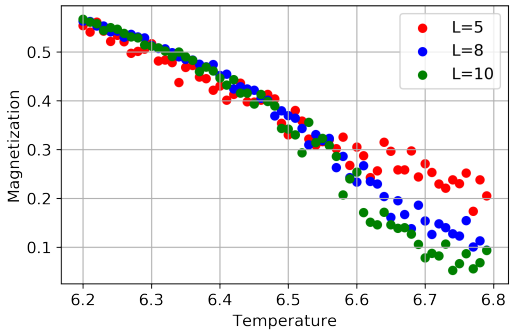


Figure 11: 4D Magnetisation

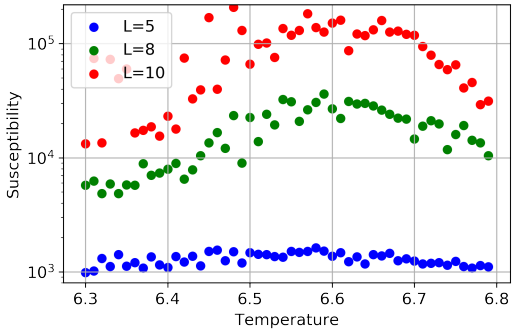


Figure 12: 4D Susceptibility

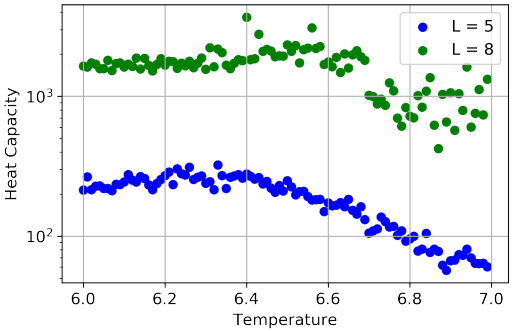


Figure 13: 4D Heat Capacity

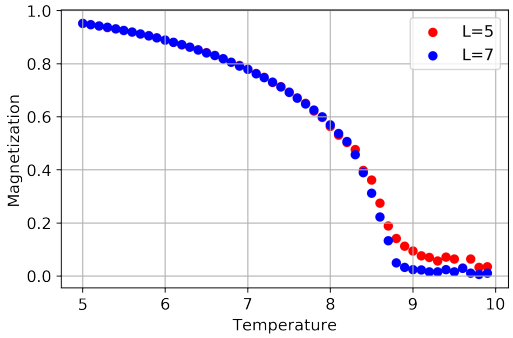


Figure 14: 5D Magnetisation

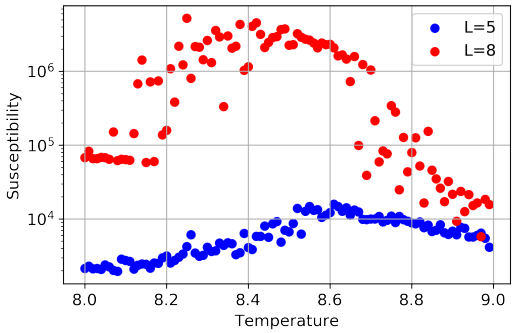


Figure 15: 5D Susceptibility

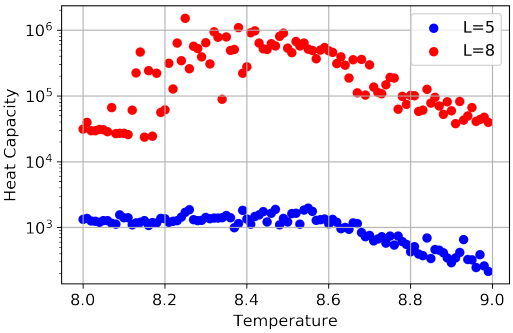


Figure 16: 5D Heat Capacity

Appendix B - Simulation Code

The entire code, including analysis and simulation, can be found on my Github. Since the majority of the code is a reiteration of the same functions just for higher dimensional lattices, here we only attach the main file and a sample class file for the 2D Lattice.

```
/* File: main.cpp - main file to run the simulation for a temperature range. */
#include <iostream>
#include "Lattice2D.h"
#include "Lattice3D.h"
#include "Lattice4D.h"
#include "Lattice5D.h"
#include <chrono>
#include <thread>
#include <vector>
#include <mutex>
#include <algorithm>
#include <functional>
#include <unordered_map>

int main() {

    auto start = std::chrono::high_resolution_clock::now();

    std::unordered_map<double, double> magnetisationData;
    std::unordered_map<double, double> m2Data;
    std::unordered_map<double, double> susceptibilityData;
    std::unordered_map<double, double> heatCapData;

    std::vector<double> magnetisations;
    std::vector<double> temperatures;
    std::vector<double> susceptibilities;
    std::vector<double> magnetisationSq;
    std::vector<double> heatCap;

    const int THERMAL = 100; // Set number of thermalising iterations
    const int nlat = 8; // Set dimension of lattice
    const size_t n_temp = 100; // Set number of temperatures to simulate over
    int counter = 0;
    const size_t n_iter = 1000; // Set number of MC iterations
    const size_t nthreads = std::thread::hardware_concurrency(); // Initialise thread
        control for parallelisation
```

```
std::vector<std::thread> threads(nthreads);
std::mutex critical;
for (int t = 0; t<nthreads; t++) {

    threads[t] = std::thread(std::bind(
            [&](const int bi, const int ei,const int t) {
               for(int i = bi; i <ei; i++) {

                   Lattice5D lat = Lattice5D(nlat); // Initialise lattice of required
                       dimension using class.

                   double temperature = i/100.0 + 8.0;
                   double magnetisationSum = 0.0;

                   double m2Sum = 0.0;

                   double energySum = 0.0;
                   double e2sum = 0.0;

                   if (temperature == 0) continue;

                   double beta = 1/temperature;
                   double J = 1.0;

                   for (int j = 0; j < n_iter; j++) {
                       lat.Wolff(J, beta); // Run Wolff algorithm for n_iter = 1000
                           iterations
                       if (j > THERMAL) {
                           double energy = lat.energy(); // If we have thermalised,
                               store values of interest
                           energySum += energy;

                           double E2 = pow(energy, 2);
                           e2sum += E2;

                           int sumOfSpins = lat.M();
                           int M = abs(sumOfSpins);
                           int M2 = pow(M, 2);
                           magnetisationSum += M;
                           m2Sum += M2;
                       }
```

```
                    }
                    std::lock_guard<std::mutex> lock(critical); // prevent parallel
                        threads from talking to save output
                    int numAvg = n_iter - THERMAL;

                    // average quantities of interest stored over the simulations
                    double avgE = energySum / numAvg;
                    double avgE2 = e2sum / numAvg;
                    double C = (avgE2 - pow(avgE, 2)) * pow(beta, 2);
                    double avgMagnet = magnetisationSum / (n_iter - THERMAL);

                    double avgM2 = m2Sum / (n_iter - THERMAL);
                    double susceptibitlity = (avgM2 - pow(avgMagnet ,2)) * beta;

                    counter++;
                    susceptibilityData[temperature] = susceptibitlity;

                    heatCapData[temperature] = C;
                    std::cout << "Done temperature " << temperature << " Counter: " <<
                        counter << std::endl;
                }
            },t*n_temp/nthreads,(t+1)==nthreads?n_temp:(t+1)*n_temp/nthreads,t));

}
std::for_each(threads.begin(), threads.end(), [](std::thread& x){x.join();});

for (auto x : heatCapData) {
    temperatures.emplace_back(x.first);
    heatCap.emplace_back(x.second);
}

for (auto x : susceptibilityData) {
    susceptibilities.emplace_back(x.second);
}

// Output temperatures with corresponding quantities of interest.

for (double temperature : temperatures) {
    std::cout << temperature << ", ";
}
std::cout << "\nHeat Capacities\n\n";
```

```
    for (double capacity : heatCap) {
        std::cout << capacity << ", ";
    }

    std::cout << "\nSusceptibilities\n\n";
    for (double sus : susceptibilities) {
        std::cout << sus << ", ";
    }

    std::cout << "\n<M>\n\n";
    for (double m : magnetisations) {
        std::cout << m << ", ";
    }

    auto stop = std::chrono::high_resolution_clock::now();

    auto duration = std::chrono::duration_cast<std::chrono::seconds>(stop - start);
    std::cout << "\nFinished in " << duration.count() << " for N = " << nlat << ",
        n_iter = " << n_iter << std::endl;

    return 0;
}
```

This is a sample file showing implementation of the key methods such as the Wolff algorithm for a 2D Lattice

---

```
//
// Created by Jyotirmai Singh on 11/17/19.
//

#include "Lattice2D.h"
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <set>
#include <unordered_set>
#include <cmath>
#include <random>
#include <stack>

Lattice2D::Lattice2D(int latSize) : Lattice(latSize) {

   for (int i = 0; i < latSize; i++) {
       std::vector<int> row;
       for (int j = 0; j < latSize; j++) {
           int spin = rand() % 2;
           if (spin == 0) spin = -1;
           row.emplace_back(spin);
       }
       _lat.emplace_back(row);
   }
}

int Lattice2D::getSite(int x, int y) {
    return _lat[x][y];
}

void Lattice2D::flipSite(int x, int y) {
    _lat[x][y] = -_lat[x][y];
}

void Lattice2D::display() {
    for (int i = 0; i < _lat.size(); i++) {
        for (int j = 0; j < _lat[i].size(); j++) {
            std::cout << _lat[i][j];
```

```
            if (j != _lat.size() - 1) {
                std::cout << ", ";
            }
        }
        std::cout << "\n";
    }
    std::cout << "\n\n";
}

struct pair_hash {
    inline std::size_t operator()(const std::pair<int,int> & v) const {
        return v.first*31+v.second;
    }
};

void Lattice2D::Wolff(double J, double b) {
    srand(time(0));
    int randX = rand() % _latSize;
    int randY = rand() % _latSize;
    std::pair<int, int> randomPoint = {randX, randY};

    std::unordered_set<std::pair<int, int>, pair_hash> cluster;
    //std::set<std::array<int, 2>> cluster;
    //std::set<std::array<int, 2>> frontier;
    std::unordered_set<std::pair<int, int>, pair_hash> frontier;
    cluster.insert(randomPoint);
    frontier.insert(randomPoint);

    double p = 1 - exp(-2*b*J);

    while (!frontier.empty()) {
        std::unordered_set<std::pair<int, int>, pair_hash> newFrontier;
        //std::set<std::array<int , 2>>::iterator it;
        for (auto point : frontier) {
            int x = point.first;
            int y = point.second;
            int siteSpin = getSite(x, y);

            std::vector<std::pair<int, int>> neighbours = getNeighbours(x, y);
            for (auto neighbour : neighbours) {
                if (!isValidCoord(neighbour)) continue;
```

```
                int neighbourX = neighbour.first;
                int neighbourY = neighbour.second;
                int neighbourSpin = getSite(neighbourX, neighbourY);

                auto iter = cluster.find(neighbour);

                // Making random number to do the r < 1-exp(2bJ) test
                std::random_device rd;
                std::mt19937 gen(rd());
                std::uniform_real_distribution<> distr(0.0, 1.0);
                double randomNum = distr(gen);
                bool probabilityTest = randomNum < p;
                if (siteSpin == neighbourSpin && iter == cluster.end() && probabilityTest)
                    {
                    newFrontier.insert(neighbour);
                    cluster.insert(neighbour);
                }
            }

        }
        frontier = newFrontier;
    }

    for (auto site : cluster) {
        int x = site.first;
        int y = site.second;
        flipSite(x, y);
    }

    //display();
}

std::vector<std::pair<int, int>> Lattice2D::getNeighbours(int x, int y) {

    std::pair<int, int> x_plus = {GARBAGE, GARBAGE};
    std::pair<int, int> y_plus = {GARBAGE, GARBAGE};
    std::pair<int, int> x_minus = {GARBAGE, GARBAGE};
    std::pair<int, int> y_minus = {GARBAGE, GARBAGE};


   if (x + 1 < _latSize) {
```

```
    x_plus.first = x + 1;
    x_plus.second = y;
} else {
    x_plus.first = 0;
    x_plus.second = y;
}

if (x - 1 >= 0) {
    x_minus.first = x - 1;
    x_minus.second = y;
} else {
    x_minus.first = _latSize-1;
    x_minus.second = y;
}


if (y + 1 < _latSize) {
    y_plus.first = x;
    y_plus.second = y + 1;
} else {
    y_plus.first = x;
    y_plus.second = 0;
}


if (y - 1 >= 0) {
    y_minus.first = x;
    y_minus.second = y - 1;
} else {
    y_minus.first = x;
    y_minus.second = _latSize-1;
}

std::vector<std::pair<int, int>> neighbours;

neighbours.emplace_back(x_plus);
neighbours.emplace_back(x_minus);
neighbours.emplace_back(y_plus);
neighbours.emplace_back(y_minus);

return neighbours;
```

```
}

bool Lattice2D::isValidCoord(std::pair<int, int> coord) {
    return coord.first != GARBAGE && coord.second != GARBAGE;
}

int Lattice2D::M() {
    int tot = 0;
    for (int i = 0; i < _latSize; i++) {
        for (int j = 0; j < _latSize; j++) {
            tot += _lat[i][j];
        }
    }
    return tot;
}

int Lattice2D::M2() {
    return pow(M(), 2);
}

double Lattice2D::magnetisation() {
    return abs(1/pow(_latSize, 2) * M());
}

double Lattice2D::susceptibility(double beta) {
    int m2 = 0;
    double mag = magnetisation();
    double m2avg = 1/pow(_latSize,2) * pow(mag, 2);
    return beta * (m2avg - pow(mag, 2));
}

double Lattice2D::energy() {
    double ene = 0;
    for (int x = 0; x < _latSize; x++) {
        for (int y = 0; y < _latSize; y++) {
            int site = getSite(x, y);
            std::vector<std::pair<int, int>> neighbours = getNeighbours(x, y);
            for (auto neighbour : neighbours) {
                int nb = getSite(neighbour.first, neighbour.second);
                ene += -nb*site;
            }
```

```
        }
    }
    return ene/4.0;
}
```