Indian Institute of Science Education and Research, Mohali
Department of Physical Sciences

# Metropolis algorithm and its implementation on 2D Ising model

Saurabh Annadate
MS16016


06-06-2020

# Contents

# 1. Summary

This project explores the Metropolis algorithm and its implementation on 2D Ising model in great detail. Monte Carlo methods are discussed to understand the similarities of interactions in the computer model of spins and the systems in real life.

The goal of this project is to explore the scope of the Metropolis algorithm in understanding various aspects of phase transitions. Also, the limitations and drawbacks of the Metropolis algorithm near the critical region are discussed. Working of Wolff algorithm to overcome such issues is also discussed in the project.

Simulation of finite size scaling is implemented to obtain the critical exponents.

# 2. Introduction

Statistical mechanics is used to calculate the properties of condensed matter system. Since the systems contains large number of atoms or molecules it is difficult to solve its enormous number of Hamiltonian equations governed by the system in order to solve the system. Therefore, statistical mechanics takes the probabilistic approach to solve these problems. For large systems, the range of all the reasonable behaviours fall into a narrow range; that allows us to state the behaviour of the real system with high confidence within that range.

## 2.1 Statistical mechanics review

The Hamiltonian systems we generally study have another component, which is the thermal reservoir. It acts as a constant source and sink of heat for our system. In effect, the reservoir acts as a perturbation on our Hamiltonian. The effect of this entire system(system+reservoir) can be understood using the dynamics, a set of rules by which system goes from one state to other.

Suppose our system is in a state $\mu$, there is a finite probability, $R(\mu \to \nu)dt$, for transition from state $\mu$ to state $\nu$. There will be transition rates for all possible paths one state to other system can take. And these transition rates are usually all we know about the dynamics. So even if we only know the starting state of our system, we know after short amount of time it can be in one of the large number of possible states. And here, probabilistic nature of our problem comes in. Then, we define weights $w_\mu(t)$ associated with the probability that system will be in the state $\mu$ at time t. We can write a master equation for the evolution of $w_\mu(t)$ in terms of the rates $R(\mu \to \nu)dt$ as following:

$$\frac{dw_\mu}{dt} = \sum_\nu [w_\nu(t)R(\nu \to \mu) - w_\mu(t)R(\mu \to \nu)][2] \tag{2.1}$$

The probabilities $w_\mu(t)$ must follow the sum rule,

$$\sum_\mu w_\mu(t) = 1 \tag{2.2}$$

The microscopic properties of the system relate to these weights by the following relation, defined as expectation at any given time,

$$\langle Q \rangle = \sum_{\mu} Q_{\mu} w_{\mu}(t) \tag{2.3}$$

Right had side of the equation 2.1 becomes zero when both the terms cancel each other out for all $\mu$, then we say, equilibrium is achieved. We will be simulating equilibrium systems using Monte Carlo techniques through out the project.

## 2.2 The idea of Monte Carlo simulation

The Monte Carlo simulation has been used to calculate the partition function of the Ising model. It simulates the random thermal fluctuations of the system from state to state. For calculating the microscopic properties we take the expectation value as a time average over all the states system go through while simulation. In a complete Monte Carlo calculation, we create a model on our computer and make it pass through a variety of steps in such a way that the probability of finding that system in that particular state at that time t is equal to $w_{\mu}(t)$ which that system would have in a real life.

We will see that, how the rates $R(\mu \to \nu)$ for transition of the system from one state to another state will be related to the equilibrium of the system with respect to Boltzmann distribution. Then the measurements of the observable quantities will be done using these equilibrium states.

The name Monte Carlo methods is coined by Nicolas Metropolis. The method is known for more than 100 years by the name statistical sampling; it was used for estimating integrals which were hard and could not be calculated by other techniques.

# 3. The Monte Carlo Method

In any thermodynamic system, the main goal is to calculate the expectation value of any observable quantity Q. We can calculate it by averaging over all the states of the system, weighing each state with its Boltzmann probability

$$\langle Q \rangle = \frac{\sum_\mu Q_\mu e^{-\beta E_\mu}}{\sum_\mu e^{-\beta E_\mu}} \tag{3.1}$$

for large system it is difficult to keep track of all the state. Monte Carlo techniques work by choosing a subset of states by random from some probability distribution and then estimate the expectation. So it is pivotal to sample the important states from large number of states and, this technique is call important sampling.

## 3.1 Important sampling

A thermodynamic process we see in real life is also a sort of analogue to Monte Carlo method. We generally see a states of system which are most probable based on Boltzmann distribution. So it only make sense to sample those states with high probabilities or in other words Boltzmann weights in our model.

## 3.2 Markov processes

To generate the appropriate set of random states according to the Boltzmann distribution Monte Carlo method rely on Markov processes. Given a system in state $\mu$ Markov process generates randomly a new state $\nu$ for the system, with a transition probability $P(\mu \to \nu)$. Transition probabilities should satisfy two conditions: (1) they should not change over time, and (2)they should depend only on the properties of the current states $\mu$ and $\nu$, and not on any other states the system has passed through. It must also satisfy the constraint

$$\sum_\mu P(\mu \to \nu) = 1 \tag{3.2}$$

## 3.3 Ergodicity

If we run our Markov process for long enough time, we should be able to reach any state starting from any other state, this is the condition of ergodicity. We will keep most of the transition probabilities to zero in our algorithm but making sure that there exists at least one path between two states with non-zero transition probability.

## 3.4 Detailed balance

Condition of detailed balance is placed on Markov processes to make sure that the system after coming to equilibrium is generated by Boltzmann distribution than any other distribution. The rate at which the system goes to a state $\nu$ from a state $\mu$ should be equal to the rate it will go the any other state $\mu$ again. Which can be written as

$$\sum_{\mu} p_{\mu} P(\mu \to \nu) = \sum_{\mu} p_{\nu} P(\nu \to \mu) \tag{3.3}$$

using the equation 3.2, we can simplify this to

$$p_{\mu} = \sum_{\mu} p_{\nu} P(\nu \to \mu) \tag{3.4}$$

For any set of transition probabilities satisfying this equation, the probability distribution $p_{\mu}$ will be an equilibrium of the dynamics of the Markov process.

## 3.5 Acceptance ratios

In Monte Carlo algorithm we can have transition which doesn't go anywhere, with $P(\mu \to \mu) = 1$. This satisfies the condition of detailed balance and gives us little more freedom over choosing other states. We can break the transition probability into two parts

$$P(\nu \to \mu) = g(\nu \to \mu) A(\nu \to \mu) \tag{3.5}$$

The quantity $g(\nu \to \mu)$ is the selection probability; it is the probability of generating state $\mu$ when the system is in state $\nu$. $A(\mu \to \nu)$ is the acceptance ratio. Ideally, the acceptance ratios should be closer to one in order to make transitions from one state to other.

# 4. The Metropolis algorithm

Metropolis algorithm is the most preliminary algorithm of the Monte Carlo methods. Here the ideas of the metropolis algorithm are discussed with the help of the Ising model. Ising model is a simple model of up and down spins on a finite lattice. The system of N spins can take $2^N$ states, and energy of any particular state is given by

$$H = -J \sum_{\langle ij \rangle} s_i s_j - B \sum_i s_i, \tag{4.1}$$

where $J$ is an interaction energy between two nearest-neighbour spins, and $B$ is an external magnetic field. But in actual simulation the external magnetic field is kept zero for simplicity of the model, and most of the phenomenon concerning Ising model can be carried out without external magnetic field.

Now, we will see how the algorithm is arrived at, it satisfies all the conditions we discussed above, in 3. Metropolis algorithm has single-spin-flip dynamics; it flips only one spin at a time. This ensures that the states will differ by small amount of energy depending on how much bonds it did brake while flipping. In our case of 2D lattice the energy difference would be $8J$ since there are four neighbouring spins, and each bond contributes $2J$ energy change. The single-spin-flip dynamics also ensures that our algorithm will follow the principle of ergodicity because the system can go to any state from any state by flipping single spin at a time. In the metropolis algorithm the selection probabilities from one state to any other possible state are equal. Suppose the system is in state $\mu$ now there are N possible spin flips for the system which would lead to N different states, $\nu$, for the system. Given that we can say the selection probability will be

$$g(\mu \to \nu) = \frac{1}{N} \tag{4.2}$$

With these selection probabilities, the condition of detailed balance take the form[2]

$$\frac{P(\mu \to \nu)}{P(\nu \to \mu)} = \frac{g(\mu \to \nu)A(\mu \to \nu)}{g(\nu \to \mu)A(\nu \to \mu)} = \frac{A(\mu \to \nu)}{A(\nu \to \mu)} = e^{-\beta(E_\nu - E_\mu)} \tag{4.3}$$

This equation puts a constraint on the acceptance ratios. In order to maximise the acceptance ratio, we will set one of them to 1 and adjust the other to satisfy equation 4.3. Suppose, $\mu$

and $\nu$ are the two states, and $E_\mu < E_\nu$. Then $A(\nu \to \mu)$ will be larger of the two acceptance ratio and would take the value of 1. It will lead to $A(\mu \to \nu)$ equal to $e^{-\beta(E_\nu - E_\mu)}$.

$$A(\mu \to \nu) = \begin{cases} e^{-\beta(E_\nu - E_\mu)}, & \text{if } E_\nu - E_\mu > 0 \\ 1, & \text{otherwise.} \end{cases} \tag{4.4}$$

The equation 4.4 makes this algorithm a metropolis algorithm. In the simple Monte Carlo studies of statistical mechanical models this algorithm has become the algorithm of choice. To summarize the algorithm:

1. Define a lattice and initialise a starting state,$E_\mu$
2. Choose a site randomly
3. Calculate the energy change if that site is flipped $E_\nu - E_\mu$
4. if $E_\nu - E_\mu < 0$, overturn the spin
5. Otherwise, Generate a random number $r$ such that $0 < r < 1$
6. If $r < e^{-\beta(E_\nu - E_\mu)}$, flip the spin
7. Repeat steps 2 to 6

# 5. Implementation of the Metropolis algorithm

In this section we will see an actual implementation of the Metropolis algorithm on $100 \times 100$ lattice Ising model. For the entire simulation we have kept the value of $J = 1$ and $B = 0$, these are the parameters from equation 4.1. firstly, we will define a lattice using 2-D array of size N=100. We will apply periodic boundary conditions to the array so that the spins on the edge will be surrounded by the spins which are on the opposite edge.

Now we will choose some starting value for each spin in the system; there are two commonly used starting states. one corresponds to $T = 0$, where all the spins points up or down(in our simulation we will keep all the spin in upward direction in this case). The second corresponds to $T = \infty$, all the spins points randomly in up or down direction in uncorrelated fashion. There is one more initial state in which the last state of nearby temperature is used as the initial state when we look at the model in varying temperature.

After initializing the starting state we choose one spin randomly from the lattice and calculate the energy difference it would make after flipping. In order to do that we first calculate the energy of the starting state $E_\mu$, then we see the neighbouring sites of the spin we want to flip because each bond break will contribute $\pm 2J$ energy depending on its own orientation. Then we will sum over all the nearest neighbours and subtract in from $E_\mu$ to get $E_\nu$(Energy of the new state).

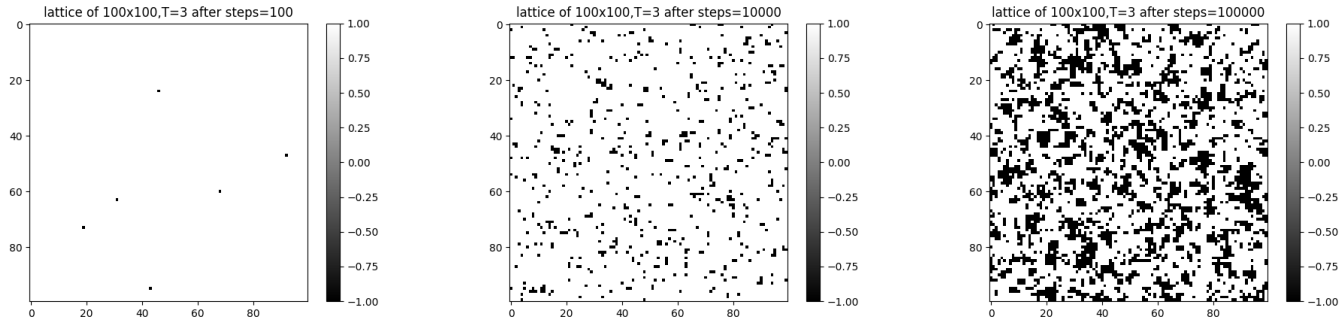
$$E_\nu - E_\mu = 2Js_k^\mu \sum_i s_i^\mu \tag{5.1}$$

where $s^\mu$ are the spins of the $\mu$ state, $i$ runs is the nearest neighbour subset, and $k$ in the randomly chosen spin. Then as given in above section we will proceed to the next steps of the algorithm. In our calculations throughout the simulation we calculate temperature in energy units, so that $k = 1$. At each cycle we will store the total energy and total magnetization of the lattice.

## 5.1 Measurements

After systems attains equilibrium, we will measure the quantities like magnetization and energy. Calculation of magnetization is easy, the total magnetization of the system in state $\mu$ is given as

$$M_\mu = \sum_i s_i^\mu \tag{5.2}$$




(a) (b) (c)

Figure 5.1: snapshots of $100 \times 100$ Ising model with $J = 1$, $B = 0$, and $T = 3$.The black dots represent down spin and white dots represent up spin. It is a visualization of how states look at metropolis steps 100,10000, and 100000 respectively.

for subsequent states we can just add the difference which would happen after flipping a single spin. Suppose, we want to flip $k^{th}$ spin, it would lead to the difference of $2s_k$ in total magnetization. So instead of summing over all the spins again we can just add the difference to get the magnetization of the new state

$$M_\nu = M_\mu + 2s_k^\nu \tag{5.3}$$

Similarly for measurements of energy we first calculate the energy of the initial state $E\mu$ like discussed in 4, then we check the energy difference $\Delta E$ which will occur if the $k^{th}$ spin is flipped. If metropolis algorithm allows that spin flip, we update the energy of the new state by

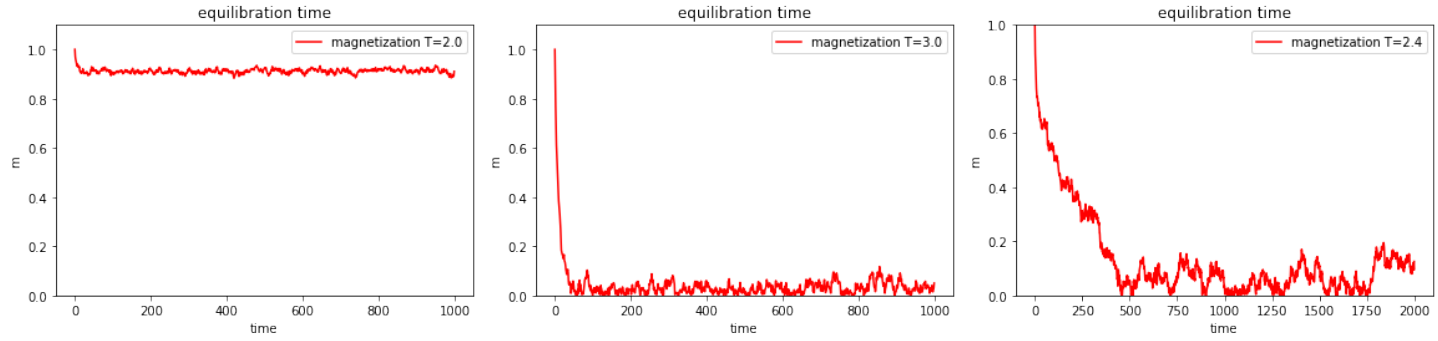
$$E_\nu = E_\mu + \Delta E \tag{5.4}$$

If we divide these quantities by total number of sites, we will get magnetization per site and internal energy.

We will also calculate the derived quantities like the specific heat and the magnetic susceptibility.

## 5.2 Equilibration time

In order to take measurements which we are interested in we will have to wait until our system comes in the states where the spins are distributed according to the Boltzmann distribution. Starting from the initial state to reach the equilibrium system takes finite amount of time which is called as equilibration time, $\tau_{eq}$. In other words, at equilibrium the average probability of finding our system in any particular state is proportional to the Boltzmann weight of that




(a) (b) (c)

Figure 5.2: Equilibration plot of a $100 \times 100$ Ising model with $J = 1$, and $B = 0$. The system has started at $T = 0$ and cooled to various different temperatures. Time has been calculated in metropolis steps per lattice site. (a) System is cooled down to $T = 2.0$, it quickly goes to equilibrium, (b) System is cooled down to $T = 3.0$, (c) System is cooled down to $T = 2.4$, it takes more time to come to equilibrium, and the values of $m$ takes wider range.

state. System spends most of the time in the equilibrium states, and it takes the values of magnetization and energy in very narrow range.

## 5.3 Correlation time

We saw in previous sections that it is important to let the system come to the equilibrium so that we can take measurements of the quantities of interest. But it is not very satisfactory to tell just by seeing at the equilibration plot, what we really interested in is the correlation time $\tau$ of the simulation. Correlation time is the measure of time taken by the system to go to the state which is completely different from the start state.

In our simulation we will calculate the correlation time by calculating the time-displaced autocorrelation function on magnetization. Autocorrelation gives a measure of correlation in quantities measured at two different times. Autocorrelation will be high at the beginning and will go to zero near the end. It is expected to fall off exponentially; we will fit it using the following equation:

$$\chi(t) = e^{\frac{-t}{\tau}}. \tag{5.5}$$

So, the measurements which are apart from each other by time $\tau$ are similar to each by $1/e$ factor. Since we want to record independent measurements, we will record the measurements which are $2\tau$ apart from each other. The formula used for autocorrelation function is[2]

$$\chi(t) = \frac{1}{t_{max} - t} \sum_{t'=0}^{t_{max}-t} m(t')m(t'+t) - \frac{1}{t_{max} - t} \sum_{t'=0}^{t_{max}-t} m(t') \times \frac{1}{t_{max} - t} \sum_{t'=0}^{t_{max}-t} m(t'+t) \tag{5.6}$$

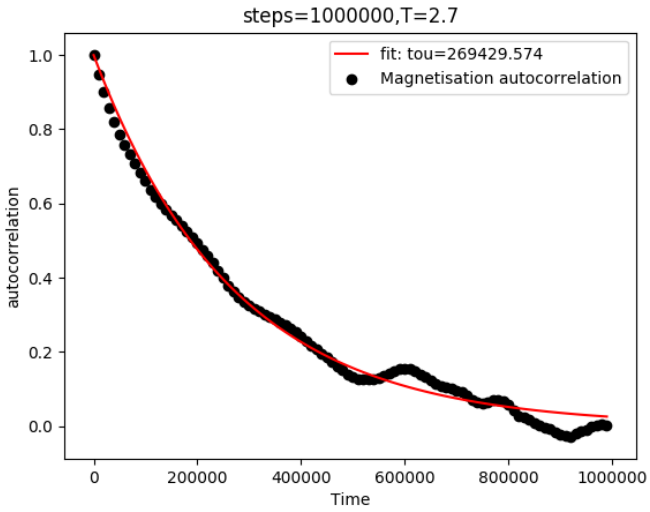



Figure 5.3: For $100 \times 100$ lattice size at temperature T=2.7, autocorrelation is plotted against time. Time is calculated in Monte Carlo steps here.

While calculating the correlation near the tail part, according to the equation 5.6, $t$ will be very close to $t_{max}$ and end up integrating over a very small value. Which will lead to larger error in $\chi$ near the tail. So we will run the algorithm for larger amount of time in actual simulation to reduce this type of error.

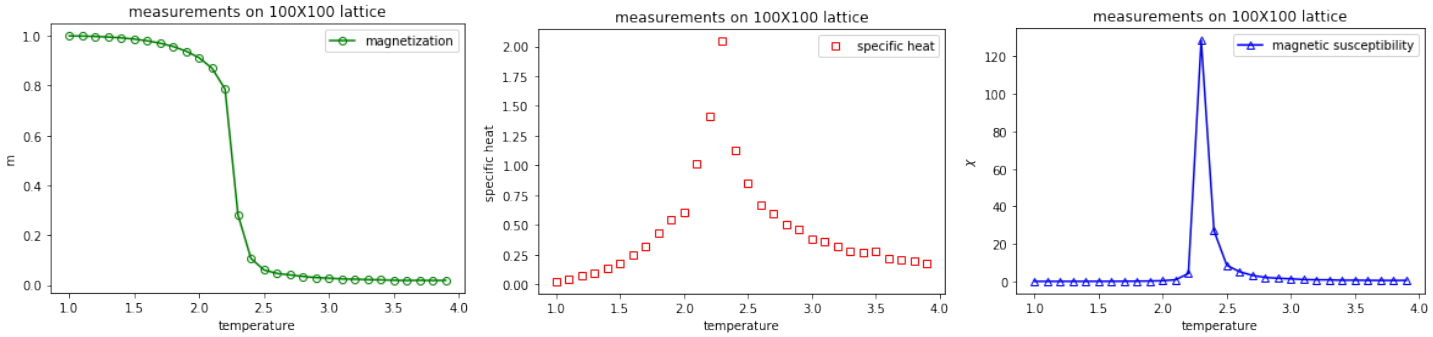
## 5.4 Calculation of thermodynamic quantities

After calculating the correlation time between two independent measurements we can easily extract the independent measurements of magnetization and energy from the data. Then we will compute the quantities like magnetization per spin, specific heat, and magnetic susceptibility, etc.

$$\langle m \rangle = \frac{1}{N} \langle \sum_i s_i \rangle \tag{5.7}$$

$$c = \frac{\beta^2}{N} (\langle E^2 \rangle - \langle E \rangle^2) \tag{5.8}$$

$$\chi = \beta N (\langle m^2 \rangle - \langle m \rangle^2) \tag{5.9}$$




(a) (b) (c)

Figure 5.4: (a)Magnetization per spin, (b)specific heat and (c)magnetic susceptibility has been plotted against temperature. Calculations are done on Ising model with $B = 0, J = 1$. Simulation is run upto $10^9$ Monte carlo steps.

## 5.5 Estimation of errors

### 5.5.1 Statistical errors

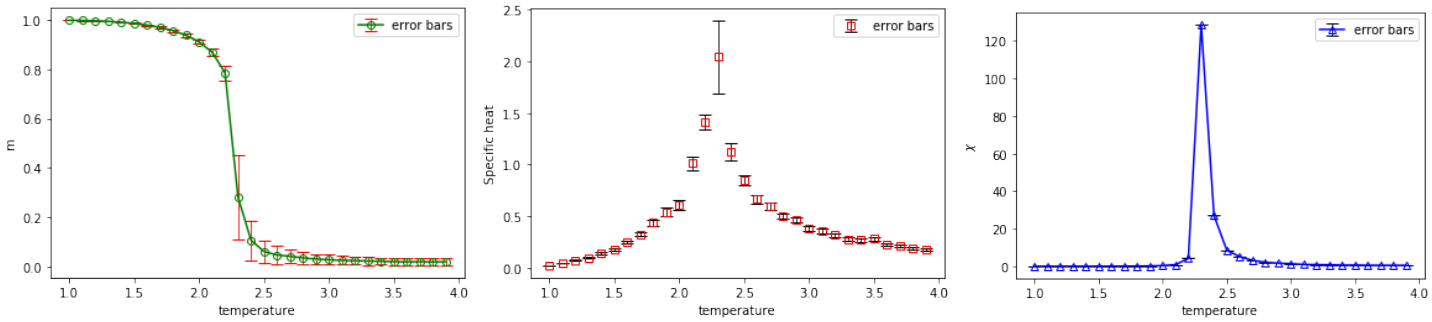
Because of Monte Carlo method there is a innate statistical error associate to our measurements. Main source of the statistical errors is thermal fluctuations that system experience because of the metropolis algorithm even after coming to the equilibrium. Suppose we make $n$ measurements of the magnetization of the system, then the best estimate will of the true thermal average of the magnetization will be the mean of those $n$ measurements, and the best estimate of the statistical error will be the standard deviation on the mean given by

$$\sigma = \sqrt{\frac{\overline{m^2} - \overline{m}^2}{n-1}} \tag{5.10}$$

This expression assumes that our measurements are independent of one another.

### 5.5.2 Jackknife method

In the case of derived quantities, which are not straightforwardly calculated at each step, we apply a different approach to calculate errors. Quantities like specific heat is not measured at each step of the simulation rather it is calculated using the averages of $E$ and $E^2$ over a long interval of time. Suppose we want to calculate the error in specific heat using Jackknife method then we will need $n$ independent measurements of energy. Using $n$ measurements we first calculate a value $c$ for specific heat. Then we calculate $n$ other estimates $c_i$ as follows: we remove the first measurements and calculate $c_1$ using the subset of other $n-1$ measurements.




(a) (b) (c)

Figure 5.5: Same quantities from previous figure are plotted with error bars on them. Error bars are calculated using above mentioned methods. Error bars are bigger near critical temperature.

Then we put the first one back and remove the second measurement, and calculate $c_2$ using the subset of the measurements, and so forth. The estimate of the error in c is

$$\sigma = \sqrt{\sum_{i=1}^{n}(c_i - c)^2}. \tag{5.11}$$

## 5.6 Optimization of the python code

The entire coding is done in Python because it is very intuitive and easy to understand language. But, python is slower than C because it is an interpreted language. That increases the number of actual CPU instructions required to perform the task. The Python code is optimized using Numba[3].Numba translates Python functions to optimized machine code at runtime using the industry-standard LLVM compiler library. Numba-compiled numerical algorithms in Python can approach the speeds of C or FORTRAN.

## 5.7 Collect measurements for different lattice sizes

The same procedure has been repeated for different lattice sizes and measurements are stored for further analysis(like data collapse and critical exponents). Measurements of Lattice sizes L=50, L=64, L=100 and L=120 are stored. In the next section we will apply finite size scaling and try to obtain critical exponents from it.

# 6. Finite size scaling

In finite size scaling we look at how the thermodynamic quantities vary with size L of the system. FSS is also helpful in giving the estimates of critical exponents. Suppose we want to do FSS for magnetic susceptibility, we know the following relations,

$$\chi \sim |t|^{-\gamma}, \xi \sim |t|^{-\nu} \tag{6.1}$$

where,

$$t = \frac{T - T_c}{T_c} \tag{6.2}$$

after eliminating t from above equations we get,

$$\chi \sim \xi^{\frac{\gamma}{\nu}} \tag{6.3}$$

Since we are dealing with finite size systems correlation length will cut off and similarly susceptibility will cut off as it approaches system size. Cut off takes place when $\xi$ approaches L, for $\xi << L$, $\chi$ takes the value same as it would take in infinite system size. So we can express the relation in following form,

$$\chi = \xi^{\frac{\gamma}{\nu}} \chi_0(L/\xi) \tag{6.4}$$

where $\chi_0$ is a dimensionless function of a single variable which takes values

$$\chi_0(x) = \begin{cases} \text{constant}, & x >> 1 \\ \sim x^{\frac{\gamma}{\nu}}, & x \to 0 \end{cases} \tag{6.5}$$

equation 6.4 has $\xi$ in it, so to eliminate it and make the equation more convenient we will define a new dimensionless function $\widetilde{\chi}$ such that:

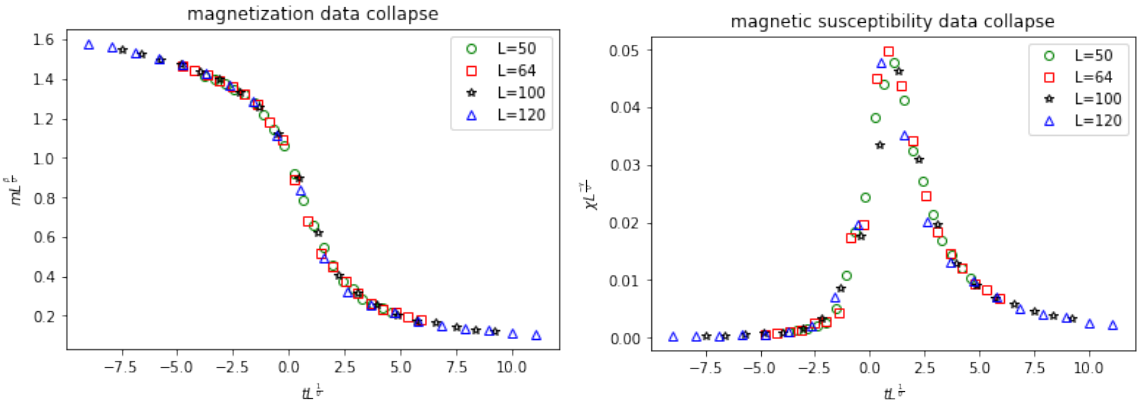
$$\widetilde{\chi}(x) = x^{-\gamma} \chi_0(x^{\nu}) \tag{6.6}$$

making use of 6.1 we can come up with basic equation for finite size behaviour of susceptibility,

$$\chi = L^{\frac{\gamma}{\nu}} \widetilde{\chi}(L^{1/\nu} t) \tag{6.7}$$

We can rearrange it as,

$$\widetilde{\chi}(L^{1/\nu} t) = L^{\frac{-\gamma}{\nu}} \chi_L(t) \tag{6.8}$$




(a) (b)

Figure 6.1: Data collapse of magnetization and susceptibility for 2D Ising model. Data points are belong to measurements of different lattice sizes as indicated. From the collapse we find $\beta = 0.125$, $\nu = 1.02$, $\gamma = 1.76$, and $T_c = 2.27J$.

and using similar arguments we can also derive the finite scaling equation for magnetization, which can be written as:[1]

$$\widetilde{m}(L^{1/\nu}t) = L^{\frac{\beta}{\nu}} m_L(t) \tag{6.9}$$

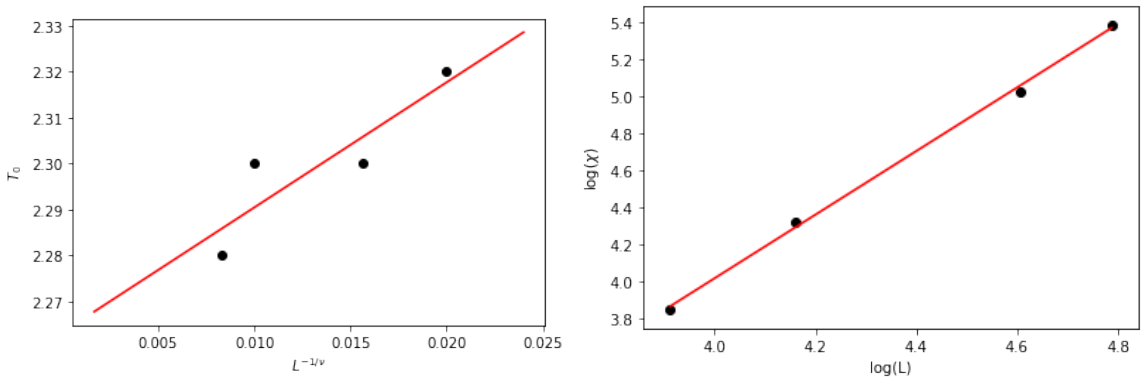
## 6.1 Critical exponents

Using the above method we did the data collapse as shown in the figure above. That collapse happened to certain critical exponents which were independent of the system sizes. The above method helped us find $\beta = 0.125$, $\nu = 1.02$, $\gamma = 1.76$, and $T_c = 2.27J$. They are closely in agreement with the real exponents which are, $\beta = 1/8$, $\nu = 1.0$, $\gamma = 7/4$, and $T_c = 2.269J$. It is difficult to calculate the error bars using this technique. We can calculate errors by estimating the region over which the collapse appears optimal(by eye). We can also use the following techniques to find errors on some of the exponents.

## 6.2 Error bars on critical exponents

We can also perform data-collapse at only one point $x_0$ at which the scaling function is maximum. Using 6.2 and $x = L^{1/\nu}t$, we can write $T_0$ corresponding to $x_0$ as,

$$T_0 = T_c(1 + x_0 L^{-1/\nu}) \tag{6.10}$$




(a) (b)

Figure 6.2: fig.(a) gives $T_c = 2.26 \pm 0.15$ as intercept and, fig.(b) gives the value of $\gamma/\nu = 1.72 \pm 0.14$ as slope.

Thus, if we plot $T_0$ against $L^{-1/\nu}$, all points should lie on a straight line, provided correct value of $\nu$, and intercept should give $T_c$. Fitting the line using least square method we can get errors on the slope and intercept. We can use bootstrap or Jackknife method to get the error on $\nu$. I have used Jackknife method and fiddled with the value of $\nu$ each time to minimize the variance while fitting the data to the straight line. By doing the above calculations I got $T_c = 2.26 \pm 0.15$ and $\nu = 1.01 \pm 0.12$.

Since the scaling function is independent of system sizes, we can write the value of susceptibility at $x_0$(at highest point) in the form,

$$\chi_L(T_0) \propto L^{\gamma/\nu}, \tag{6.11}$$

Thus, if we plot the maximum value of $\chi$ against $L$ on logarithmic scale, we should get a straight line and its slope should give the value of $\gamma/\nu$. Again, after fitting it we will get the error on $\gamma/\nu$, and since we know the error on $\nu$ we get the error on $\gamma$. After fitting the the graph I got $\gamma/\nu = 1.72 \pm 0.14$ as slope and we already know $\nu = 1.01 \pm 0.12$, which gives us $\gamma = 1.73 \pm 0.26$.

# 7. Critical slowdown

We are mostly curious about the properties of the Ising model around the critical temperature $T_c$, above this temperature the system is in paramagnetic phase, and Below it the system is in ferromagnetic state. The magnetization is referred as the order parameter of the Ising model. At higher temperature the spins tends to be uncorrelated to each other but as we lower the temperature of the system spins start to form clusters and align with each other. At $T_c$, the correlation length $\xi$ diverges, and we can see large clusters of the spins pointing up or down. Below $T_c$ these clusters spontaneously choose to have most of its spin in up or down direction given net magnetization. These clusters also contribute to magnetization and energy significantly, as they flip from one orientation to other they produce critical fluctuations. As we go near $T_c$, the correlation length $\xi$ diverges which lead to bigger fluctuations in $m$ and $E$. Since the quantities like specific heat and magnetic susceptibility depends on these fluctuations, they also diverge at $T_c$.

Metropolis algorithm is least accurate and slow in this region because of the two reasons. The first reason is **critical fluctuations**, the statistical errors because of the large fluctuations near $T_c$ are large. We can take more independent measurements to reduce the errors which will incorporate near $T_c$. But one more difficulty occurs while handling the previous one, the correlation time also diverges as we go near $T_c$. The large correlation time results in larger computation time for more independent measurements.

Critical fluctuations is an innate property of the Ising model, any Monte Carlo algorithm will have high fluctuations in the critical region. However, the large correlation time is a drawback of the Metropolis algorithm which is called as **critical slowdown**. To overcome this problem we will try another Monte Carlo algorithm in the next section.

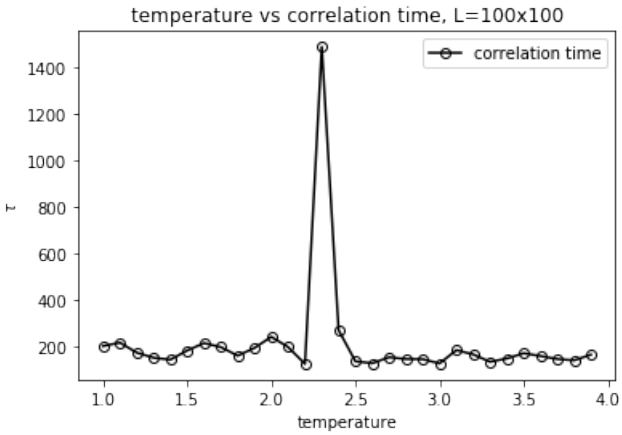



Figure 7.1: Correlation time for 2D Ising model for $J = 1, B = 0$. Correlation time is calulated in Monte Carlo steps per lattice site. We see huge increase in $\tau$ near the critical region.

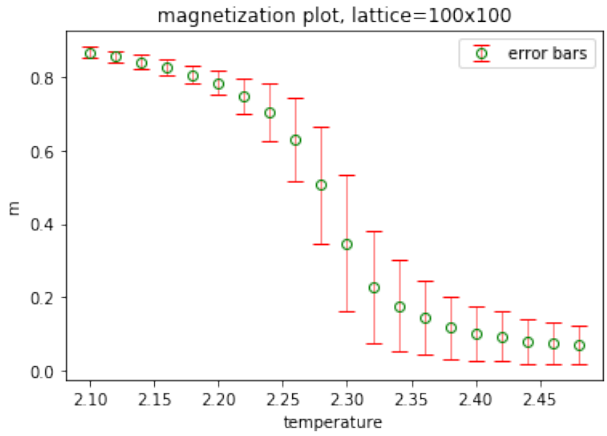



Figure 7.2: magnetization per site for 2D Ising model for $J = 1, B = 0$. The measurements are mainly done in near critical region, it shows large error bars in this very region because of critical fluctuations.

# 8. Wolff Algorithm: A cluster flipping algorithm

We know that Metropolis algorithm is slow near critical temperature, it is mainly because near $T_c$ domain formation happens and it is difficult for metropolis algorithm to flip such a large domains by single flip at a time. The probability of flipping a spin which is surrounded by other similarly oriented spins becomes really low so the algorithm becomes very slow.

To overcome this problem Ulli Wolff proposed a cluster flipping algorithm is now known as Wolff algorithm. The algorithm is as follows:

1. Choose a seed spin at random from the lattice.
2. Look in turn at each of the neighbours of that spin. If they are pointing in the same direction as the seed spin, add them to the cluster with probability $P_{add} = 1 - e^{-2\beta J}$.
3. For each spin that was added in the last step, examine each of its neighbours to find the ones which are pointing in the same direction and add each of them to the cluster with the same probability $P_{add}$.
4. As the cluster becomes larger some of the neighbour spins may have been considered for the addition to the cluster, but rejected. These spins will get another chance to get into the cluster.
5. repeat the above step until there is no spin left in the cluster whose neighbours are not considered for inclusion in the cluster.
6. Flip the cluster.

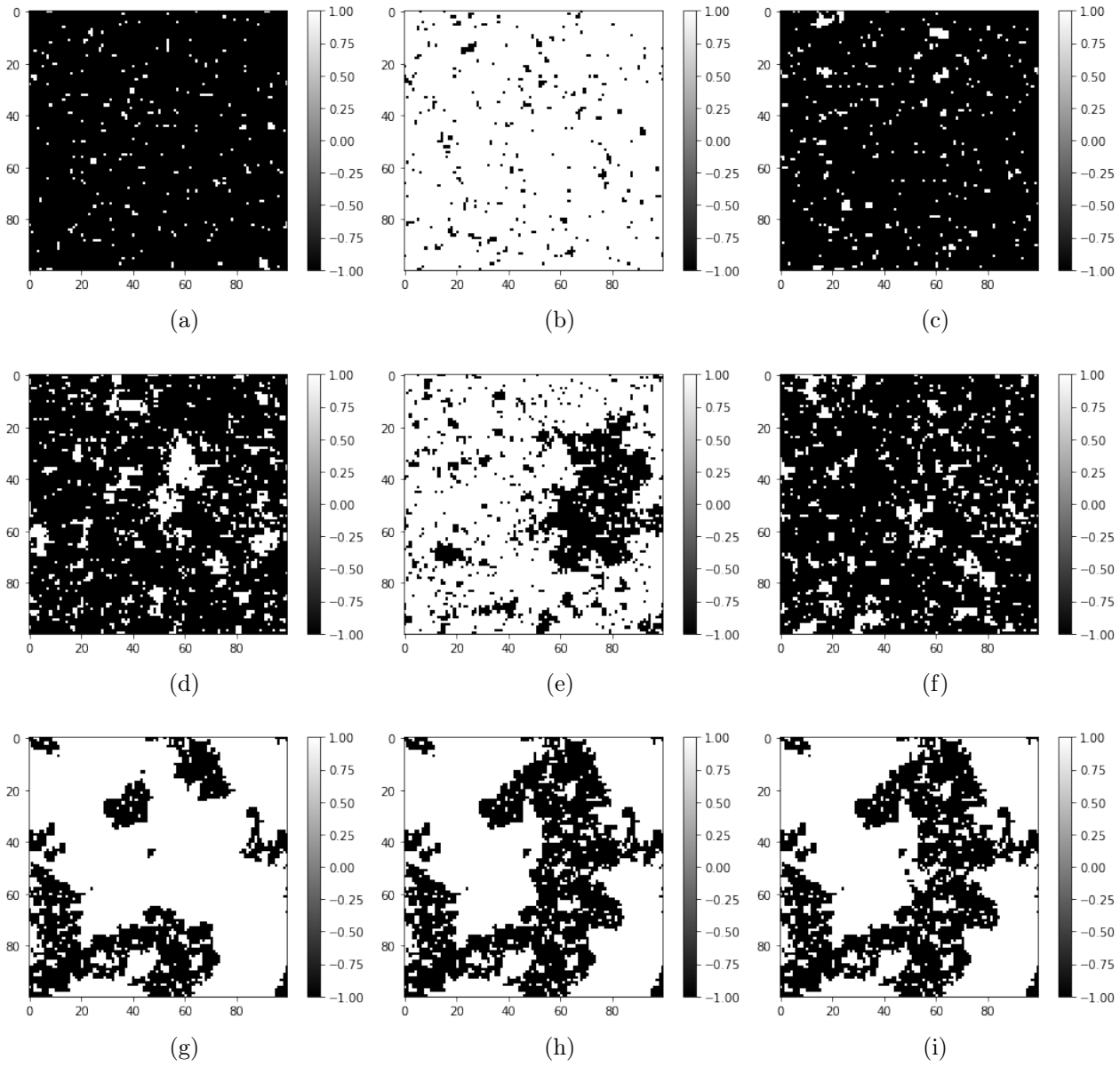



Figure 8.1: Snapshots taken on the consecutive states of $100 \times 100$ Ising model simulated over Wolff algorithm. first row from above are the states at $T = 2J$, middle row is near critical temperature at $T = 2.7J$, and the last row is at $T = 3J$. We can see that as the temperature increases the size of cluster to be flipped decreases.

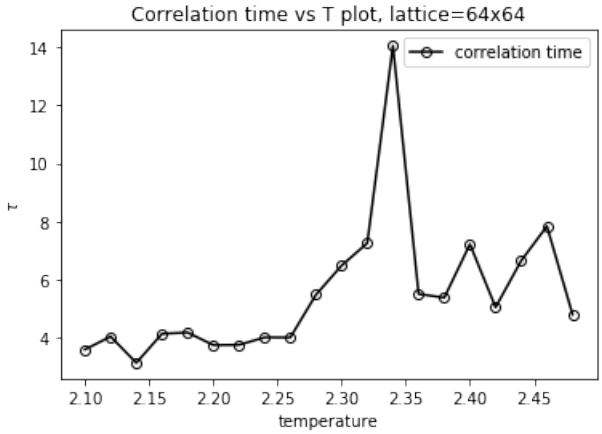
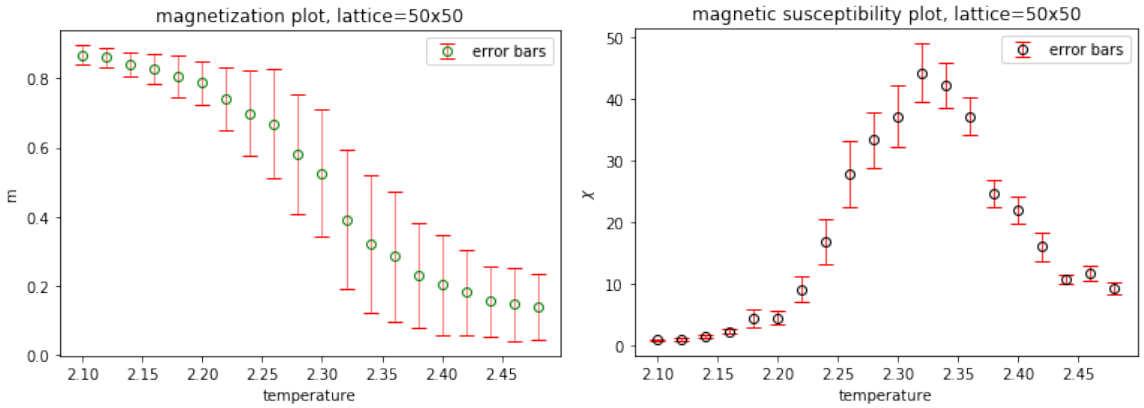



Figure 8.2: Correlation time for $64 \times 64$ Ising model simulated over Wolff algorithm. The correlation times are dropped largely compared to the Metropolis algorithm for all temperatures.




(a) (b)

Figure 8.3: Measurements of magnetization and magnetic susceptibility near critical region on $50 \times 50$ Ising model simulated over Wolff algorithm.

# 9. Results

- The Metropolis algorithm is implemented on the Ising model successfully.
- The simulation demonstrated the easily observable phase transitions at critical temperature in magnetization, magnetic susceptibility, and specific heat.
- Critical slowdown is observed in the critical region while simulating Metropolis algorithm.
- The critical exponents obtained agree closely with the literature values. I found the values of critical exponents as follows: $\beta = 0.125$, $\nu = 1.01 \pm 0.12$, $\gamma = 1.73 \pm 0.26$, and $T_c = 2.26J \pm 0.15J$.
- Wolff clustering algorithm is helpful in solving the problem of critical slowdown.

# 10. Conclusions

Metropolis algorithm is an effective and most preliminary algorithm of the Monte Carlo methods. It can be implemented quickly using Python to study the phase transitions on 2D Ising model. Numba can heavily optimize and accelerate the python code so that we can run it for larger Monte Carlo steps. Various thermodynamic quantities like magnetization, specific heat, and magnetic susceptibility shows the predicted behaviour. The critical exponents calculated using finite size scaling are in agreement with literature. Metropolis algorithm slows down near the critical temperatures. Wolff algorithm can be useful in critical region.

# Bibliography

[1] David P. Landau and Kurt Binder. *A Guide to Monte-Carlo Simulations in Statistical Physics*. Cambridge University press, The Edinburgh Building, Cambridge CB2 8RU, UK, 2009.

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

[3] Numba. Numba makes python code fast `http://numba.pydata.org`.

# Appendix : Python Programs

## Metropolis algorithm

```
def metropolis2(N,steps,lattice,T):
    E = TotalEnergy(lattice,N)
    M = TotalMagnetisation(lattice)
    Elist = []          #to store Energy at each time-step
    mlist = []          #to store Magnetization
    n = 0
    for i in range(steps):
        x = random.randrange(N)     # random x cordinate
        y = random.randrange(N)     # random y cordinate
        dE = -2* Energy_site(N,lattice,x,y)   #energy change
        if (dE < 0):
            lattice[x][y] *= -1
            E += dE
            M += 2*lattice[x][y]
            if n%(N*N) == 0:
                Elist.append(E)
                mlist.append(abs(M)/(N*N))
        elif random.random() < np.exp(-1.0 * dE/T):
            lattice[x][y] *= -1
            E += dE
            M += 2*lattice[x][y]
            if n%(N*N) == 0:
                Elist.append(E)
                mlist.append(abs(M)/(N*N))
        else:
            if n%(N*N) == 0:
                Elist.append(E)
                mlist.append(abs(M)/(N*N))
        n += 1
```

```
    return lattice,mlist,Elist
```

## Autocorrelation

```
# the function takes the input of mlist, tmax=maximum time for which
#magnetization has been recorded and t1 = interval
#(at what interval we want to get correlation between 2 measurements)
def autocorrelation(m,tmax,t1):
    corr = []
    time =[]
    for t in np.arange(1,tmax,t1):
        term1 = 0
        term2 = 0
        term3 = 0
        coefficient = 1/(tmax-t)
        for tdash in range(tmax-t):
            term1 += m[tdash]*m[tdash+t]
            term2 += m[tdash]
            term3 += m[tdash+t]
        corr.append(coefficient*term1-
                            coefficient*term2*coefficient*term3 )
        time.append(t)
    return time,corr

def func(x,tou):    #fitting function
    return np.exp(-x/tou)
```

## Jackniffe method

```
 #this function takes the measurements of energy/mag at
 given temperature and lattice size; returns error
def jackknife(Elist,N):      #N = lattice size
    errors = []
    for i,t in zip(Elist,temperature):
        c_values = []
        c = SpecificHeat(i,t,N)
        for j in range(len(i)):
            temp = i.pop(0)
            c_values.append(SpecificHeat(i,t,N))
            i.append(temp)
        sigma_square = [(k-c)*(k-c) for k in c_values]
```

```
        sigma = np.sqrt(np.sum(sigma_square))
        errors.append(sigma)
    return errors
```

# Wolff Algorithm

```
def neighbour_sites(x,y,N):
    neighbours = {(x,y):  [((x + 1) % N,y),((x - 1 + N) % N,y),
                  (x,(y + 1) % N),(x,(y - 1 + N) % N)]}
    return neighbours

def wolff_algorithm(N,steps,lattice,T):
    E = TotalEnergy(lattice,N)
    M = TotalMagnetisation(lattice)
    Elist = []          #to store Energy at each time
    mlist = []          #to store Magnetization at each time
    p_add =  1.0 - np.exp(-2.0 / T)
    for step in range(steps):
        x = random.randrange(N)     #random x cordinate
        y = random.randrange(N)     #random y cordinate
        boundary_spins, Cluster = [(x,y)], [(x,y)]
        while boundary_spins != []:
            j = random.choice(boundary_spins)
            neighbours = neighbour_sites(*j,N)
            for l in neighbours[j]:
                if lattice[l] == lattice[j] and l not in Cluster
                and random.uniform(0.0, 1.0) < p_add:
                    boundary_spins.append(l)
                    Cluster.append(l)
            boundary_spins.remove(j)
        for j in Cluster:
            lattice[j] *= -1
        E = TotalEnergy(lattice,N)
        M = TotalMagnetisation(lattice)
        Elist.append(E) #appending energy per site
        mlist.append(abs(M/(N*N)))
    return lattice,mlist,Elist
```