CBE 641: Nano Transport

Project Report

# Ising Model

(Coding experience: I have written $\sim 300 - 400$ pages of code before, primarily in python and matlab. However, I decided to do this project in a different programming language, namely Julia, for learning purposes.)

Kshitiz Parihar

April 30, 2022

# Contents

# List of Figures

# 1 Introduction

Ising model, a simple statistical physics model that exhibits phase transition, was first proposed by William Lenz in 1920, and its one dimensional case was exactly solved in 1924 by his student Ernst Ising (however, there is no phase transition in 1D Ising model) [1]. Phase transition for higher dimensional Ising models has been extensively studied, and exact analytical solution for 2D Ising model in the absence of external magnetic field was found by Lars Onsager in 1944 [1].

An example of phase transition where Ising model has been widely used is magnetism. In a magnetic material, each of the atoms have an associated magnetic spin. Depending on whether the spins are aligned or randomly distributed, the material can have a net magnetic moment (ferromagnetic) or no magnetic moment (paramagnetic). In the absence of an external magnetic field, paramagnetism is observed at high temperatures and ferromagnetism at low temperature. The critical temperature for this magnetic phase transition is called Curie temperature ($T_c$).

In this project, phase transition for 2D Ising model was studied numerically using Monte Carlo simulations performed via two different algorithms, namely Metropolis and Wolff. Autocorrelation times, critical slowing down phenomena, spin-spin correlation function and finite size scaling were investigated for the 2D lattice system. Phase transition characteristics for a 3D Ising model were also evaluated.

# 2 Methodology

## 2.1 2D Ising Model

In this study, our system of interest is $L$ x $L$ square lattice with each $(i,j)^{th}$ lattice site having spin ($s_{ij}$) with only two possible orientation, i.e. $s_{ij} \in \{-1, +1\}$ (See Fig. 2.1). The Hamiltonian for the system in the absence of external magnetic field is given by

$$\mathcal{H} = -\frac{J}{2} \sum_{(i,j)} s_{i,j} \left( s_{i-1,j} + s_{i+1,j} + s_{i,j-1} + s_{i,j+1} \right) \tag{1}$$

where $J$ is the interaction strength of neighboring spins. Note that divide by 2 is to prevent double counting. To avoid treating the boundaries of the lattice differently, periodic boundary conditions are imposed. Magnetization of the system is defined as

$$M = \sum_{(i,j)} s_{i,j} \tag{2}$$

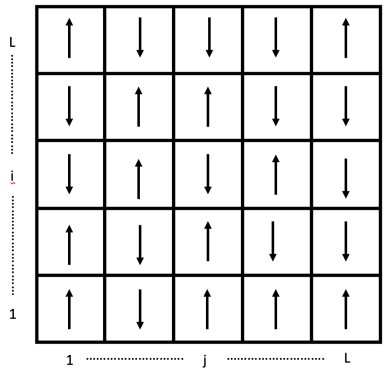


**Figure 2.1:** 2D Ising model: square lattice where each site has either spin up ($\uparrow \equiv +1$) or spin down ($\downarrow \equiv -1$)

Note that for all the Monte Carlo simulations performed in this study on the Ising model, $J$ and $k_B$ were taken as equal to 1.

## 2.2 Metropolis Monte Carlo

The Metropolis algorithm is a widely used for performing Monte Carlo simulations for a system in equilibrium at constant temperature. Metropolis Monte Carlo (MMC) involves generating a set of Boltzmann weighted spin configurations. For Ising model, MMC can be implemented as follows:

1. Generate an initial configuration of spins. Generally, either a "uniform" configuration where all spins are same or "random" configuration where each spin has equal probability of being $\pm 1$ are used as initial microstates.

2. Select a spin at random. Calculate the change in system energy ($\Delta\mathcal{H}$) if the spin selected in step 2 is flipped.

3. Update rule (probability of flipping the spin)

$$\mathcal{W}(s_{ij} \to -s_{ij}) = \begin{cases} 1 & \Delta\mathcal{H} \leq 0 \\ e^{-\frac{\Delta\mathcal{H}}{k_B T}} & otherwise \end{cases} \tag{3}$$

Note that one Monte Carlo step (MCS) (or sweep) refers to $L^2$ repetition of steps 2 and 3 (i.e. on average all spins in the lattice are chosen once for flipping in each MCS). The update rule in MMC (3) ensures that *detailed balance* is satisfied. Julia code used for performing Metropolis Monte Carlo on 2D Ising model is provided in Appendix A.1 and A.2.

## 2.3 Wolff Algorithm

An important limiting factor for Metropolis algorithm is the phenomena of *critical slowing down* near the critical point (discussed further in section 3.2). This computational bottleneck of single spin flip-based Metropolis algorithm can be overcome by using cluster flipping-based algorithms such as Wolff algorithm [2]. For Ising model, Wolff algorithm can be implemented as follows:

1. Generate an initial configuration of spins.
2. Pick a spin at random (seed). Store its orientation and flip it.
3. Growing the cluster of spins to be flipped:
   (a) For each of the neighboring spins, if the orientation is same as the host, flip it with probability p. *Detailed balance* is satisfied for (see [3] for details)
   $$p = 1 - e^{-2J/k_B T} \tag{4}$$
   (b) Repeat (a) with the flipped spin as the new seed.

Note that each Monte Carlo Step in Wolff algorithm based Monte Carlo involves flipping a cluster of spins simultaneously. Julia code used for performing Wolff algorithm based Monte Carlo on 2D Ising model is given in Appendix A.3 and A.4.

## 2.4 Observables

The expectation value for an observable from the Monte Carlo simulation is given by

$$\langle O \rangle = \frac{1}{N} \sum_{i=1}^{N} O_i \tag{5}$$

where $N$ are number of configurations sampled from the simulation and $O_i$ is the value of the observable for the $i^{th}$ sampled configuration. Magnetization per spin ($\langle |m| \rangle$)

$$\langle |m| \rangle = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{M}{L^2} \right| \tag{6}$$

serves as the order parameter (an observable to determine the phase of the system) for the 2D Ising model. Mgnetic susceptibility ($\mathcal{X}$) and specific heat ($C_v$) for the Ising model can be evaluate using [2]

$$\mathcal{X} = \frac{1}{L^2} \frac{(\langle \mathcal{M}^2 \rangle - \langle |\mathcal{M}| \rangle^2)}{k_B T} \tag{7}$$

$$C_v = \frac{1}{L^2} \frac{(\langle \mathcal{H}^2 \rangle - \langle \mathcal{H} \rangle^2)}{k_B T^2} \tag{8}$$

Note that division by $L^2$ in both the expressions is for normalizing per spin.

# 3 Results and Discussion

## 3.1 Thermalization Process

In Monte Carlo simulation, certain number of initial MCS need to be discarded before we can sample system configurations for analysis to ensure that only equilibrium configuration are taken into account. This process of waiting for system to equilibirate is called thermalization.

In this section we analyze how the number of Monte Carlo steps required for thermalization for MMC depends on the initial configuration used and the system temperature. A square lattice of size $L = 100$ with a representative temperature in each of the three temperature regions ($T < T_c$ (aligned spins); near critical $T_c$ (clusters of spins); $T > T_c$ (randomly distributed spins)) was used. Magnetization and energy per spin for the system were used to characterize thermalization. Some key observations are as follows:

- For low temperature $T < T_c$, compared to uniform initial spin configuration, starting with a random configuration increases the thermalization time considerably (Fig. 3.1(a)). This is due to aligned spins being the preferred state of the system at low temperatures.
- Near the critical temperature, for both random and uniform initialization, the magnetization fluctuates heavily and it is not easy to determine when equilibrium is reached (Fig. 3.1(b)). Interestingly, energy per spin stabilizes quickly.
- For high temperature $T > T_c$, sweeps required to thermalize are not much different between random and uniform initializations (Fig. 3.1(c)).

Considering that "uniform" initialization performs better or similarly to "random" initialization for different values of temperature, "uniform" spin configuration was chosen as the default initial system state for all the Monte Carlo simulations performed hereafter. This analysis also gives us rough estimates for the MCS to be discarded at the beginning of each MMC simulation for the three temperature ranges. Taking a cautious approach in choosing number of MCS steps to ensure the system equilibirates

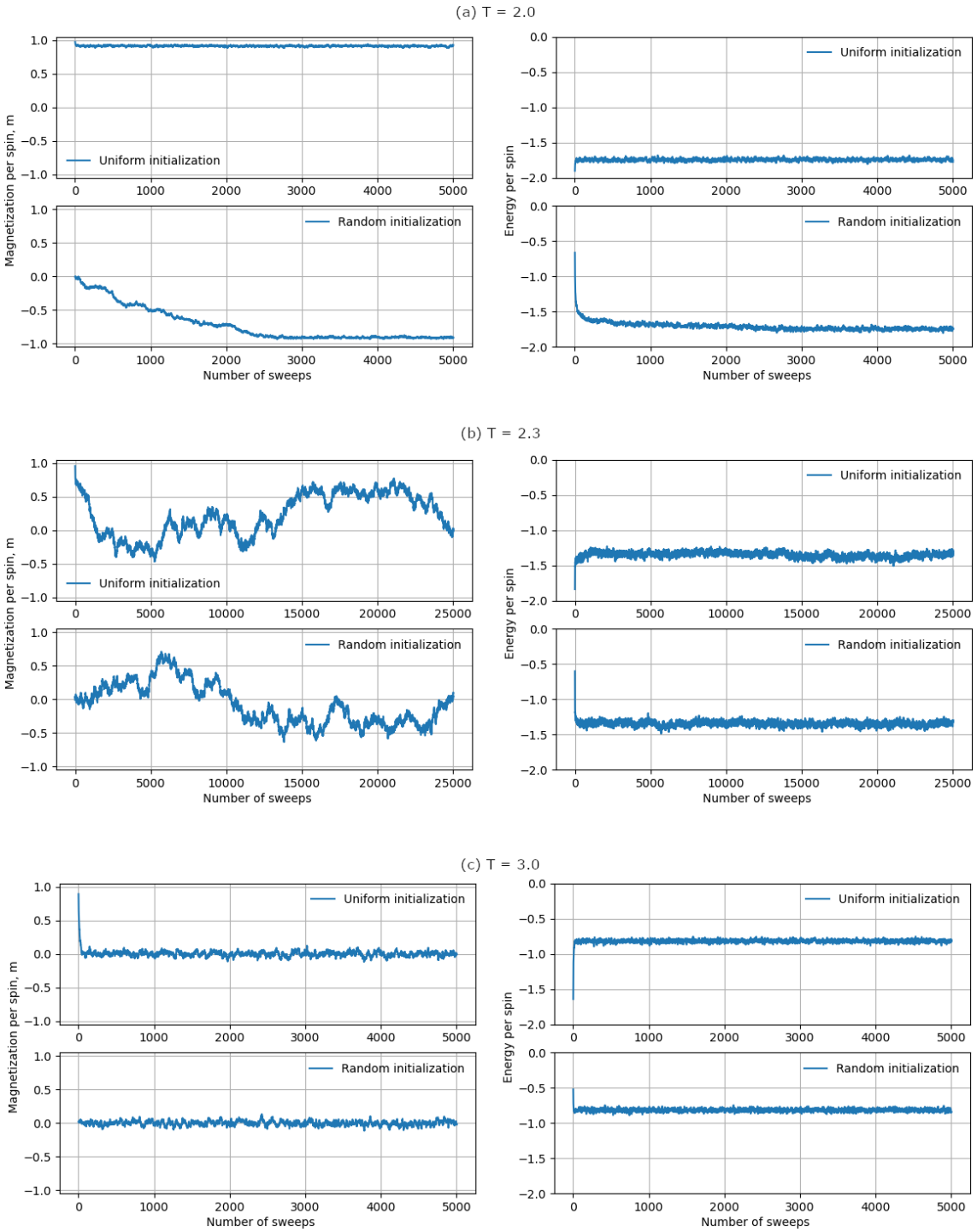


**Figure 3.1:** Metropolis Monte Carlo trajectories for (a) $T < T_c$, (b) $T$ near $T_c$, and (c) $T > T_c$. Note that for $T$ in critical region, number of MCS are much higher.

in MMC, especially close to the critical temperature,

- $T \leq 2$ and $T \geq 2.6$: Sample after 500 MCS
- $2.0 < T < 2.25$ and $2.35 < T < 2.6$: Sample after 5000 MCS
- $2.25 \leq T \leq 2.35$: Sample after 10000 MCS

## 3.2 Autocorrelation time

### 3.2.1 Metropolis Monte Carlo

Different observables (described in section 2.4) can be computed using configurations sampled from equilibrated data in Monte Carlo simulations. To ensure that sampling time is large enough for sampled configurations to be *independent* (uncorrelated) data points, the autocorrelation time for the Monte Carlo trajectory has to be determined.

Figure 3.2(a) shows the autocorrelation plots for $|m|$ at different values of T for a 100 x 100 square lattice simulated using MMC. Assuming the autocorrelation time ($\tau$) to be when the value for the autocorrelation falls below $e^{-1}$, we can see $\tau$ increases rapidly near the critical temperature. As shown in Fig. 3.2(b), the value for $\tau$ diverges as we approach the critical temperature along with higher correlation time observed for larger lattice sizes. This phenomena of diverging autocorrelation time near critical temperature is called *critical slowing down*. Due to this, significantly larger number of MCS are required for simulating a system at temperatures near the critical point using MMC. Based on these results for MMC,

- For all further measurements in this study maintain sampling time defined as $max(100, 3\tau)$

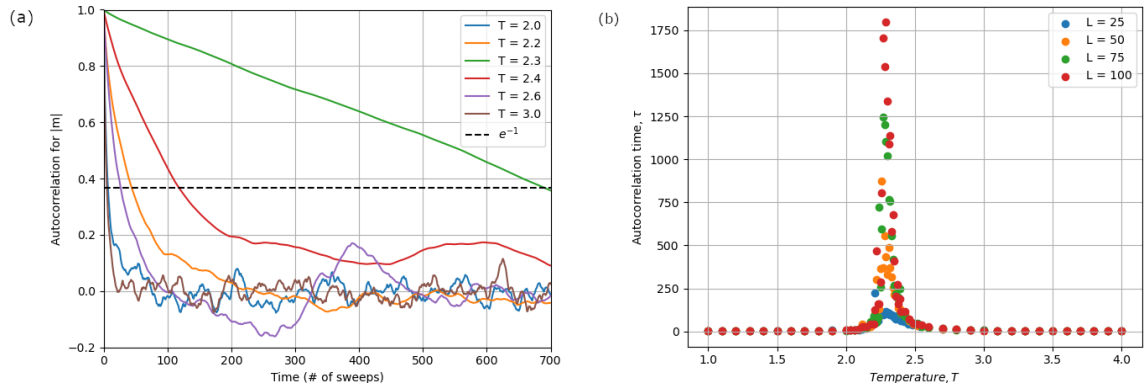


**Figure 3.2:** (a) Autocorrelation plots of $|m|$ for different $T$ values for Metropolis Monte Carlo trajectories; (b) Autocorrelation time ($\tau$) for different lattice sizes

- To ensure that sufficient number of *independent* configuration are sampled from simulation, the total number of MCS used are
  - $T \leq 2$ and $T \geq 2.6$: 5000 MCS
  - $2.0 < T < 2.25$ and $2.35 < T < 2.6$: 25000 MCS
  - $2.25 \leq T \leq 2.35$: 150000 MCS

### 3.2.2 Overcoming critical slowing down

It is expected that $\tau \propto L^z$ in the critical region for MMC, where $z \approx 2.16$ [2]. From our MMC simulations, the autocorrelation time was found to be scaling as $\tau \propto L^{2.12}$ at the critical temperature (Fig. 3.3(a)). This property of $\tau$ shows the limitation of the Metropolis algorithm when it comes to larger lattices. Since each Monte Carlo step in MMC involves $L^2$ moves, the computation time can be estimated to scale at least with $L^4$ leading to computations slowing down significantly for larger lattices.

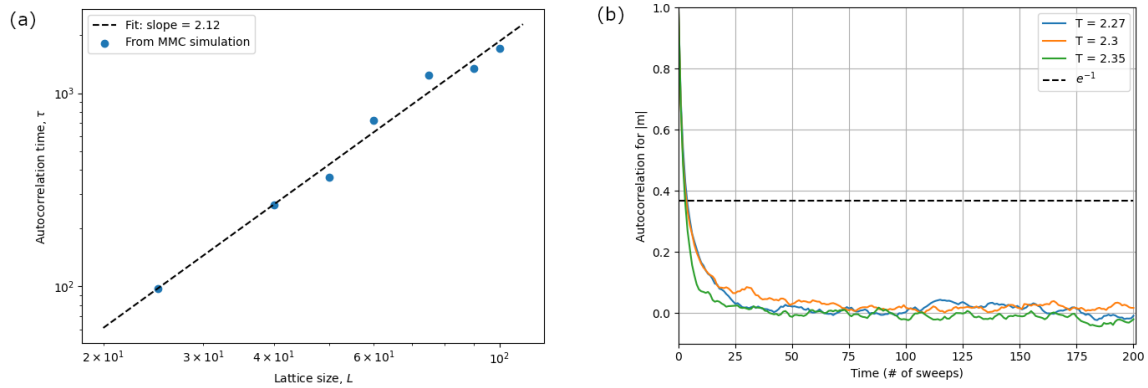


**Figure 3.3:** (a) Scaling of autocorrelation time ($\tau$) with lattice size ($L$) for Metropolis Monte Carlo at critical temperature, $T_c = 2.269$; (b) Autocorrelation plots of $|m|$ for different $T$ values for Wolff algorithm based Monte Carlo trajectories

Computationally expansive Monte Carlo simulations for large lattices sizes in the critical region can be avoided by using Wolff algorithm (single cluster-flipping based method) instead of Metropolis algorithm. Wolff algorithm makes use of the fact that clusters of spins exist near the critical temperature and thereby, flipping cluster of spins instead of flipping single spins (as done in Metropolis algorithm) would be more efficient to obtain different equilibrium system configuration in the critical region. Figure 3.3(b) shows the autocorrelation plots for $|m|$ at different values of T for a 100 x 100 square lattice simulated using Wolff algorithm based Monte Carlo. We can see

that the autocorrelation times are much smaller than those observed for MMC for those critical region temperatures.

## 3.3 Magnetization, Susceptibility and Specific Heat

Using the results from previous sections for thermalization and autocorrelation times, the measurements for the absolute value of the magnetization $|m|$ (Eqn. 6), magnetic susceptibility $\mathcal{X}$ (Eqn. 7) and specific heat $C_v$ (Eqn. 8) were performed for different values of temperature using Metropolis Monte Carlo (Fig. 3.4). Key observations are

- Clearly, signs of a phase transition can be seen, since the magnetization increases rapidly in a region around critical temperature $T_c = 2.269$ (Fig. 3.4(a)).

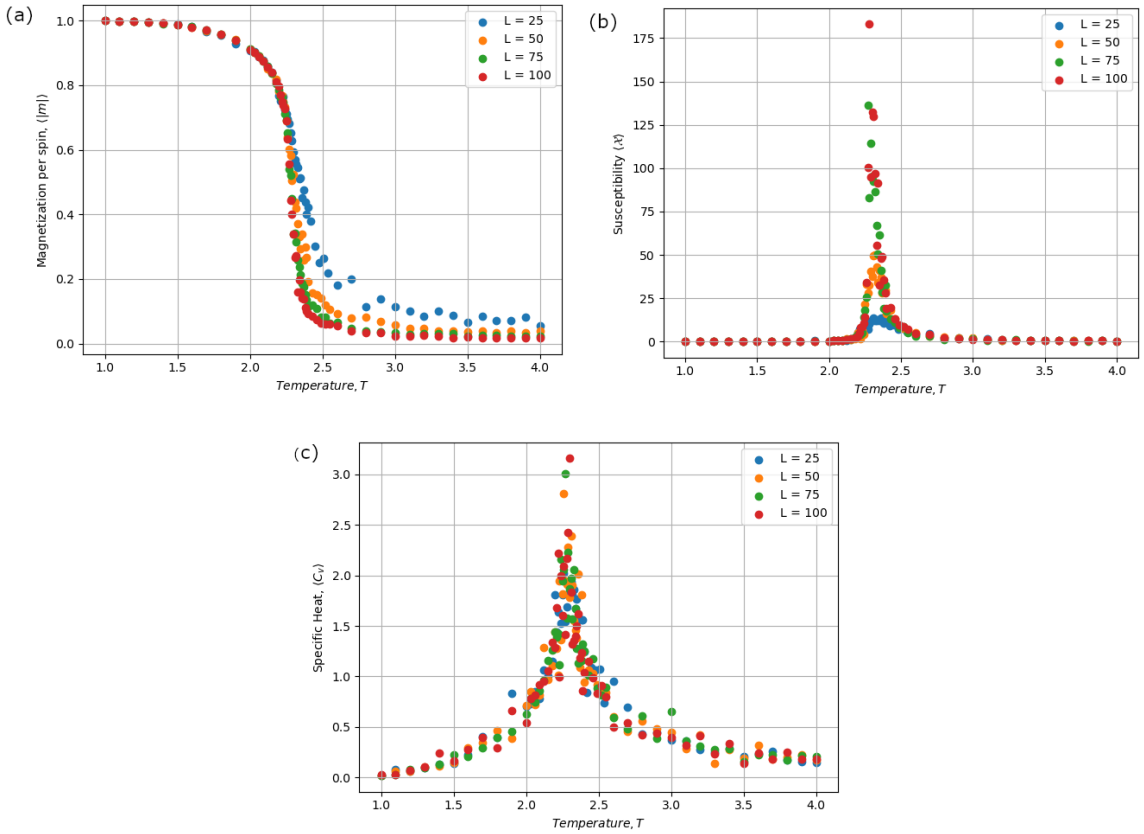


**Figure 3.4:** Using Metropolis Monte Carlo: (a) magnetization per spin $\langle|m|\rangle$, (b) magnetic susceptibility $\mathcal{X}$, and (c) specific heat $C_v$.

- Due to finite-size system considered here, the behaviour differs a bit from what the theory predicts for the thermodynamic limit. For instance, magnetization

does not fall to exactly zero for $T > T_c$. Similarly, while in the thermodynamic limit $\mathcal{X}$ and $C_v$ are expected to diverge to $\infty$ at $T_c$, the simulation results instead show $\mathcal{X}$ and $C_v$ having a maxima around $T_c$ (Fig. 3.4(b) and (c)).

- As the lattice size becomes larger, the phase transition observed becomes more and more similar to the expected transition based on analytical solution from Lars Onsager. This can be seen from steeper increase in the critical region for magnetization at larger $L$, and the maximum of $\mathcal{X}$ and $C_v$ around $T_c$ gets increasingly pronounced with higher $L$.

## 3.4 Spin-Spin Correlation Function

At low temperature, the system is dominated by a few large clusters of correlated spins. At very high temperature, the spins are essentially random. As temperature approaches toward the Tc, the critical temperature, clusters of spins start to form on all scales (up to the largest allowed by the size of the domain). In order to observe this, we calculate the spin-spin correlation function

$$g(r) = \langle s(k)s(k+r)\rangle \tag{9}$$

It was calculated for $r = 1...L/2$. Note the averaging being done: for one configuration and one value of r, average over all cells k. For simplicity, the correlation function was only calculated horizontally for each row for a particular configuration and then averaged (over all rows) to obtain $g(r)$ for the configuration. Following this, for same r, average was done over many spin configurations, each separated by many spin-flip steps (Sect. 3.2.1).

Figure 3.5 shows the correlation function for different values of $T$ where simulations are performed using MMC and lattice size used is 100. At the low temperatures, the correlation function remains close to one showing that the spin alignment is almost independent of the distance separating two spins for $T < T_c$. This changes around the critical temperature where the alignment between spins is significantly stronger over short distances than it is over long distances. However, the alignments are still present at the larger distances in the critical region. Once we get above $T_c$, the correlation function drops off rapidly after very short distances.

Correlation length ($\xi$) at a particular temperature can be calculated by using a modified version of the correlation function (i.e. $f(r) = \langle s(k)s(k+r)\rangle - \langle s(k)\rangle\langle s(k+r)\rangle$) and approximating it with an exponential decay ($e^{-r/\xi}$). Near the critical temperature $\xi$ is expected to vary with $T$ according to

$$\xi(T) \sim |T - T_c|^{-v} \tag{10}$$

where $v$ is called a critical exponent and for 2D Ising model on square lattice has value of 1.

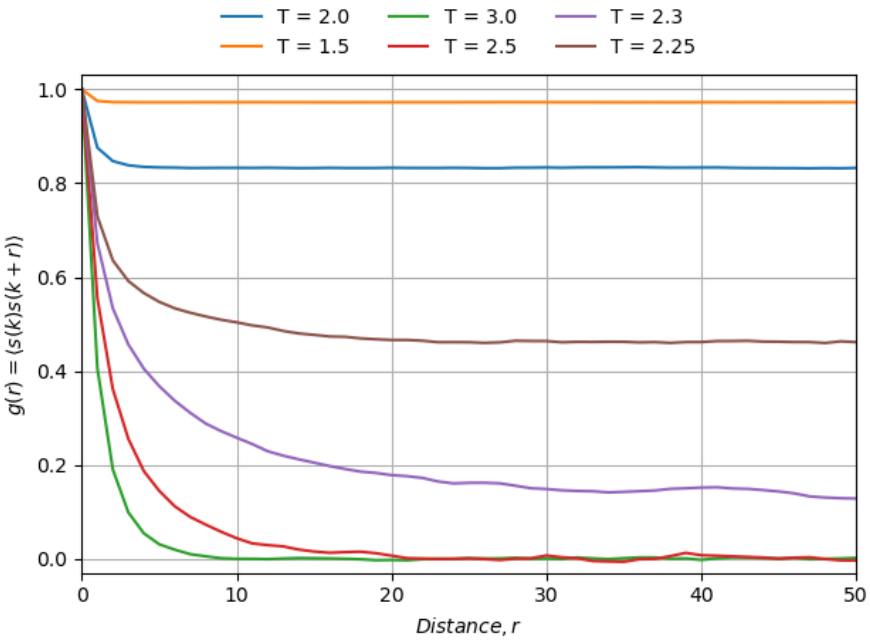


**Figure 3.5:** Spin-spin correlation function at different values of $T$ for 2D Ising model of lattice size 100.

## 3.5 Finite Size Scaling

Since we can only simulate finite size systems, we have to make use of *finite-size scaling* to extrapolate finite $L$ results to thermodynamic limit of $L \to \infty$. According to *finite-size scaling* theory, $T_c(L)$ scales as [4]

$$T_c(L) - T_c(L \to \infty) \sim aL^{-1/v} \tag{11}$$

where $a$ is a constant and $v$ is the critical exponent defined in Eqn. 10. For a finite size system, we can thus expect the correlation length $\xi$ to be comparable with the size of the system at the thermodynamic limit critical temperature $T_c(L \to \infty)$. Therefor, we can write

$$\xi(T_c(L \to \infty)) \sim |T_c(L \to \infty) - T_c(L)|^{-v} \sim L \tag{12}$$

The behavior of magnetization $|m|$ and magnetic susceptibility $\mathcal{X}$ near the critical temperature can be defined as [4]

$$m(T) \sim (T_c - T)^{\beta} \tag{13}$$

$$\mathcal{X}(T) \sim |T - T_c|^{-\gamma} \tag{14}$$

using critical exponents $\beta$ and $\gamma$ respectively. Using the similar analogy as done in Eqn. 12 where for a finite size system at critical temperature, we can write

$$m(T_c(L \to \infty)) \sim (T_c(L) - T_c(L \to \infty))^{\beta} \sim L^{-\beta/v} \tag{15}$$

$$\mathcal{X}(T_c(L \to \infty)) \sim |T_c(L \to \infty) - T_c(L)^{-\gamma} \sim L^{\gamma/v} \tag{16}$$

Therefore, based on the above equations and that $v = 1$, we can calculate $\beta$ and $\gamma$ by simulating different size systems at $T_c(L \to \infty) = 2.269$ and fitting values of $m$ and $\mathcal{X}$ values at different $L$ to Eqn. 15 and 16 respectively. Note that since, we have to simulate the system in the critical region and also need to consider large lattices, Wolff algorithm was used for performing the Monte Carlo simulations for this analysis to avoid *critical slowing down*. The sampling time used for them was 100 MCS (note that MCS in Wolff algorithm implies one randomly chose cluster flip), and samples are collected after discarding initial 3000 MCS to ensure equilibiration after "uniform" initialization. Total MCS used were 15000.

Figure 3.6 shows the fits for $\langle |m| \rangle$ and $\mathcal{X}$ at $T_c(L \to \infty)$ as a function of lattice size. The values obtained for $\beta$ and $\gamma$ from the simulations are

$$\hat{\beta} = 0.126 \pm 0.008 \tag{17}$$

$$\hat{\gamma} = 1.751 \pm 0.086 \tag{18}$$

These are consistent with values of the critical exponents reported in literature [1].

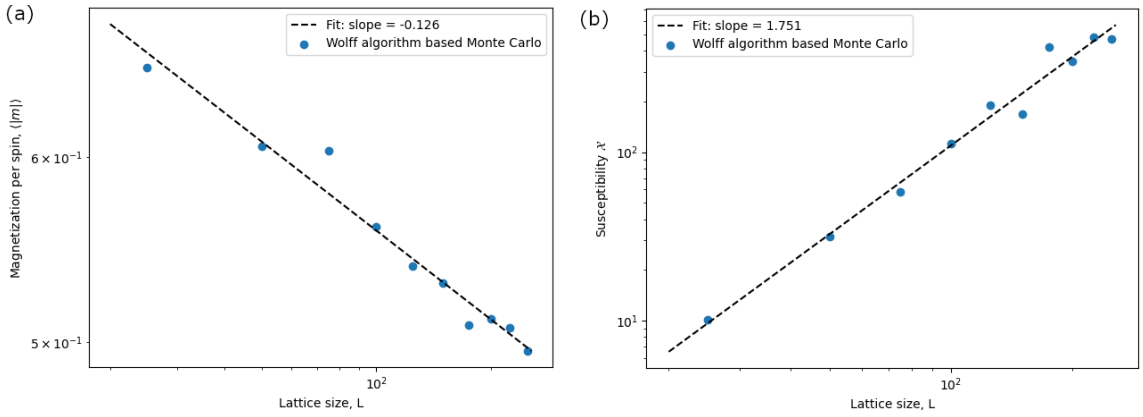


**Figure 3.6:** Scaling with lattice size for (a) magnetization per spin $\langle |m(T_c(L \to \infty))| \rangle$ and (b) magnetic susceptibility $\mathcal{X}(T_c(L \to \infty))$.

## 3.6 3D Ising Model

Three dimensional Ising model was also simulated using Metropolis Monte Carlo. The methodology for simulation is same as described in section 2.2 with the system under consideration being a 3D cube of dimension $L$ x $L$ x $L$. Similar to the analysis performed for 2D Ising in section 3.1 and 3.2.1, thermalization and autocorrelation was performed for 3D Ising. Note that the julia code used for performing Metropolis Monte Carlo on 3D Ising model is given in Appendix A.5 and A.6. The simulation parameters are as follows:

- $T \leq 4.2$ and $T \geq 4.8$: Total 10000 MCS, 1000 discarded for thermalization, sampling time $= max(100, 3\tau)$
- $4.2 < T < 4.8$: Total 50000 MCS, 5000 discarded for thermalization, sampling time $= max(100, 3\tau)$

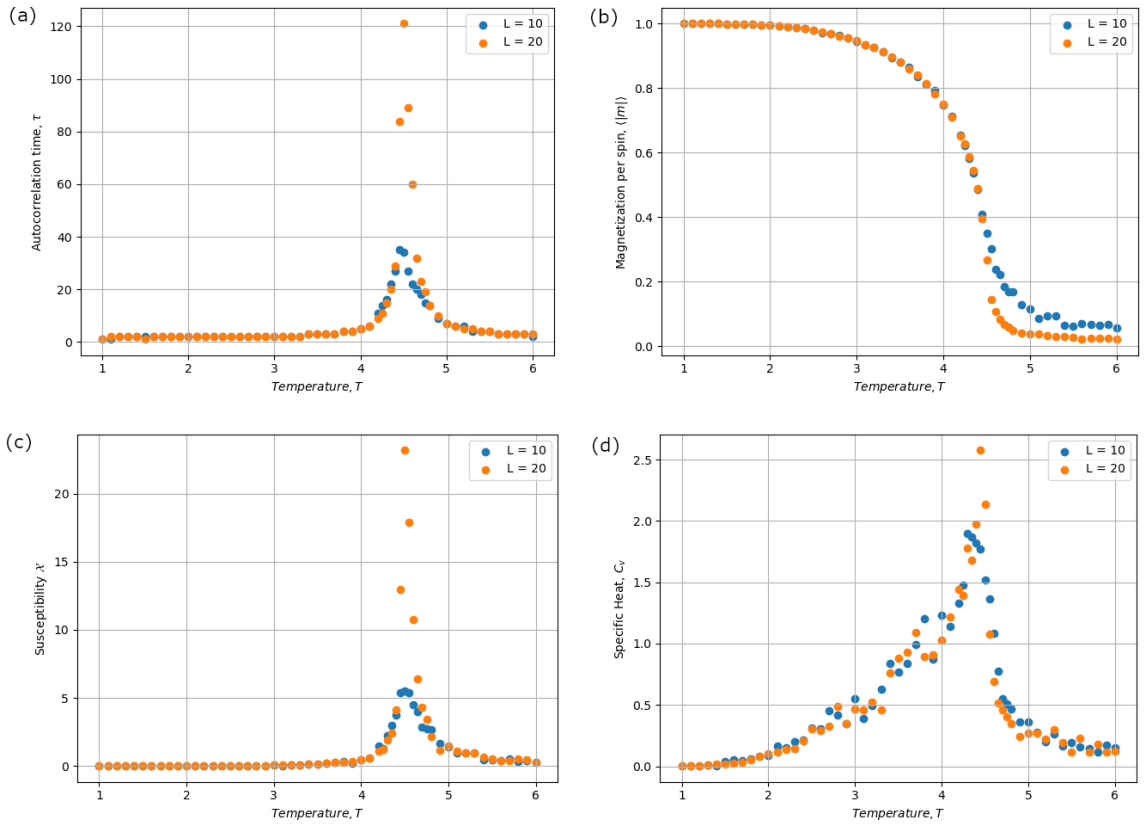


**Figure 3.7:** Using Metropolis Monte Carlo for 3D Ising model: (a) Autocorrelation time ($\tau$) for different lattice sizes, (b) magnetization per spin $\langle|m|\rangle$, (c) magnetic susceptibility $\mathcal{X}$, and (d) specific heat $C_v$.

Figure 3.7 shows the results from simulations of 3D Ising Model. We can see that similar to the 2D case, the autocorrelation time starts increasing rapidly as we approach critical temperature (Fig. 3.7(a)). Based on the transition seen for magnetization (Fig. 3.7(b)), the critical temperature for 3D Ising model seems to be $\sim 4.5$, which is consistent with reported values of $T_c$ in literature [2]. Susceptibility $\mathcal{X}$ and specific heat $C_v$ diverge and have a maximum around $T \sim 4.5$ (Fig. 3.7(c) and (d)), further demonstrating that the critical temperature for 3D Ising model is approximately 4.5 (units of $J/k_B$).

# 4 Conclusions

In this project, 2D Ising model was simulated on square lattice using Metropolis and Wolff algorithm based Monte Carlo. The phenomena of *critical slowing down* was shown for MMC along with the application of Wolff algorithm to overcome this computation limitation of MMC. Phase transition for the 2D lattice was observed based on the rapid change in magnetization near the critical temperature which was accompanied by maxima for susceptibility and specific heat. Spin-spin Correlation function for different temperature values was also analyzed to examine how the spin alignment over distances change with temperature. The critical exponents, which define how magnetization and susceptibility vary with temperature near the critical temperature, values were also determined by simulating different lattice sizes at $T = 2.269$ using Wolff algorithm based Monte Carlo. Lastly, 3D Ising model involving a cubic system was also simulated and phase transition characteristics were observed.

# References

[1] Harvey Gould and Jan Tobochnik. Magnetic systems. In *Statistical and Thermal Physics*, pages 231–293. Princeton University Press, 2010.

[2] Kurt Binder and Erik Luijten. Monte carlo tests of renormalization-group predictions for critical phenomena in ising models. *Physics Reports*, 344(4-6):179–253, 2001.

[3] Werner Krauth. Cluster monte carlo algorithms. In Alexander K Hartmann and Heiko Rieger, editors, *New Optimization Algorithms in Physics*, pages 5–22. Wiley Online Library, 2004.

[4] Harvey Gould, Jan Tobochnik, and Wolfgang Christian. Monte carlo simulations on thermal systems. In *An Introduction to Computer Simulation Methods: Applications to Physical System*, pages 582–661. 3rd edition, 2016.

# A Julia Code

## A.1 Ising_2D_metropolis.jl

```
module Ising_2D_metropolis

    using Random
    using StatsBase

    struct run_parameters
        ## Paramters for running Monte Carlo
        N_sweeps::Int64     #Total MCS
        N_skip::Int64       #Sampling time
        N_thermalize::Int64 #Steps to discard initially
    end

    mutable struct lattice
        ## Lattice parameters
        L::Int64            #Lattice size
        beta::Float64       #1/(kB*T)
        config              #storing -1/+1 for lattice sites
    end


    function initialize(init::String,L::Int)
        #=
            Initial spin configuration

            Parameter:
            lattice - struct (ising_pars)

            Return:
            config - initial lattice configuration
        =#

        if init == "uniform"
            config = ones(L,L)
        elseif init == "random"
            config = 2*bitrand((L,L)) - ones(L,L)
```

```
        end

        return config
    end


    function periodic_boundary(index::Int64,L::Int64)
        #=
            Check for boundary index

            Parameter:
            index - Int (index to be checked for boundary condition)

            Return:
            integer
        =#

        if index == 0
            return L
        elseif index == L + 1
            return 1
        else
            return index
        end
    end


    function del_energy(lat::lattice,x::Int64,y::Int64)
        #=
            Calculate the change in energy after spin flip

            Parameters:
            lat - struct lattice
            x, y - Int (indices of the spin flip)

            Return:
            e - Float  (change in energy)
        =#

        e = begin 2*lat.config[x,y]*
```

```
                (lat.config[periodic_boundary(x-1,lat.L),y] +
                 lat.config[periodic_boundary(x+1,lat.L),y] +
                 lat.config[x,periodic_boundary(y-1,lat.L)] +
                 lat.config[x,periodic_boundary(y+1,lat.L)]) end

    return e
end


function magnetization(lat::lattice)
    #=
        To calculate magnetizaton (order parameter)

        Parameters:
        lat - struct lattice
    =#
    M = sum(sum(lat.config))
    return M, M^2
end


function energy(lat::lattice)
    #=
        To calculate energy

        Parameters:
        lat - struct lattice
    =#
    E = 0
    for i in 1:lat.L
        for j in 1:lat.L
            temp = -0.5*del_energy(lat,i,j)
            E += temp
        end
    end
    E /=2
    return E, E^2
end
```

```
function run_MC(lat::lattice,p::run_parameters,corr_check::Bool)
    #=
        To run Monte Carlo

        Parameters:
        lat - struct lattice
        p - parameters for running MC
        corr_check - whether to calculate correlation function
                     or not (true or false)
    =#

    m = []      #to store magnetization
    m2 = []     #to store (magnetization)^2
    e = []      #to store energy
    e2 = []     #to store (energy)^2

    if corr_check
        corr = zeros(Int64(floor(lat.L/2))+1)
    end

    ## Start Monte Carlo
    for i in 1:p.N_sweeps
        for _ in 1:lat.L^2
            ## Choose some random spin to flip
            x, y = rand(1:lat.L,2)

            ## Change in energy due to flip
            delE = del_energy(lat,x,y)

            ## Perform Metropolis check and flip
            if delE <= 0
                lat.config[x,y] = - lat.config[x,y]
            else
                check = rand(Float64)
                if check < exp(-lat.beta*delE)
                    lat.config[x,y] = - lat.config[x,y]
                end
            end
        end
```

```
            ## Check if thermalization time is over
            ## if yes, check for sampling time
            if i >= p.N_thermalize && i%p.N_skip == 0
                temp, temp2 = magnetization(lat)
                push!(m,temp)
                push!(m2,temp2)
                temp, temp2 = energy(lat)
                push!(e,temp)
                push!(e2,temp2)
                if corr_check
                    corr += correlation(lat)
                end
            end

            ## Print to keep track of Monte Carlo steps
            if i%1000 == 0
                println("MC step: $i")
            end
        end

        if corr_check
            ## Returning magnetization per spin, susceptibility,
            ##specific heat and correlation function

            return  mean(abs.(m))/lat.L^2,
                    lat.beta*(mean(m2) - mean(abs.(m))^2)/L^2,
                    lat.beta^2*(mean(e2) - mean(e)^2)/L^2,
                    corr/size(m)[1]
        else
            ## Returning magnetization per spin, susceptibility
            ## and specific heat

            return  mean(abs.(m))/lat.L^2,
                    lat.beta*(mean(m2) - mean(abs.(m))^2)/L^2,
                    lat.beta^2*(mean(e2) - mean(e)^2)/L^2
        end
    end


    function correlation(lat::lattice)
```

```
    #=
        to calculate correlation function for 2D square lattice

            Parameters:
            lat: Struct lattice

            Returns an array with ith element corresponding to
            correlation function value at g(i-1)
    =#

    corr = zeros(Int64(floor(lat.L/2))+1)
    sample = zeros(Int64(floor(lat.L/2))+1)

    for i in 1:lat.L
        for j in 1:lat.L
            l = min(Int64(floor(lat.L/2)),lat.L-j)
            for k in j:j+l
                corr[k-j+1] += lat.config[i,j]*lat.config[i,k]
                sample[k-j+1] += 1
            end
        end
    end

    return corr./sample

end


function main(N_sweeps::Int64,N_skip::Int64,N_thermalize::Int64,
              L::Int64,init::String,T::Float64,corr_check::Bool)
    #=
        main function to perform Metropolis Monte Carlo

        Parameters:
        N_sweeps: Total number of MCS
        N_skip: Sampling time
        N_thermalize: Number of MCS to skip initially
        L: Lattice size
        init: Initial configuration to use ("uniform" or "random")
        T: Temperature
```

```
            corr_check: whether to perform correlation function or not
                        (true or false)
        =#

        p = run_parameters(N_sweeps,N_skip,N_thermalize)
        lat =  lattice(L,1/T,initialize(init,L))
        temp = run_MC(lat,p,corr_check)

        ## Return magnetization per spin, susceptibility,
        ## specific heat (and correlation function)
        return temp
    end

end
```

## A.2 main_2D_metropolis.jl

```
    include("Ising_2D_metropolis.jl")
    using .Ising_2D_metropolis

    N_sweeps = 5000
    N_skip = 100
    N_thermalize = 500
    L = 100
    init = "uniform"
    T = 2.0
    corr_check = false

    m, x, cv = Ising_2D_metropolis.main(N_sweeps, N_skip, N_thermalize,
                                        L, init, T, corr_check)
```

## A.3 Ising_2D_wolff.jl

```
module Ising_2D_metropolis

    using Random
    using StatsBase
```

```
struct run_parameters
    ## Paramters for running Monte Carlo
    N_sweeps::Int64     #Total MCS
    N_skip::Int64       #Sampling time
    N_thermalize::Int64 #Steps to discard initially
end

mutable struct lattice
    ## Lattice parameters
    L::Int64            #Lattice size
    beta::Float64       #1/(kB*T)
    config              #storing -1/+1 for lattice sites
end


function initialize(init::String,L::Int)
    #=
        Initial spin configuration

        Parameter:
        lattice - struct (ising_pars)

        Return:
        config - initial lattice configuration
    =#

    if init == "uniform"
        config = ones(L,L)
    elseif init == "random"
        config = 2*bitrand((L,L)) - ones(L,L)
    end

    return config
end


function periodic_boundary(index::Int64,L::Int64)
    #=
        Check for boundary index
```

```
        Parameter:
        index - Int (index to be checked for boundary condition)

        Return:
        integer
    =#

    if index == 0
        return L
    elseif index == L + 1
        return 1
    else
        return index
    end
end


function del_energy(lat::lattice,x::Int64,y::Int64)
    #=
        Calculate the change in energy after spin flip

        Parameters:
        lat - struct lattice
        x, y - Int (indices of the spin flip)

        Return:
        e - Float  (change in energy)
    =#

    e = begin 2*lat.config[x,y]*
              (lat.config[periodic_boundary(x-1,lat.L),y] +
               lat.config[periodic_boundary(x+1,lat.L),y] +
               lat.config[x,periodic_boundary(y-1,lat.L)] +
               lat.config[x,periodic_boundary(y+1,lat.L)]) end

    return e
end


function magnetization(lat::lattice)
```

```
    #=
        To calculate magnetizaton (order parameter)

        Parameters:
        lat - struct lattice
    =#
    M = sum(sum(lat.config))
    return M, M^2
end


function energy(lat::lattice)
    #=
        To calculate energy

        Parameters:
        lat - struct lattice
    =#
    E = 0
    for i in 1:lat.L
        for j in 1:lat.L
            temp = -0.5*del_energy(lat,i,j)
            E += temp
        end
    end
    E /=2
    return E, E^2
end


function run_MC(lat::lattice,p::run_parameters,corr_check::Bool)
    #=
        To run Monte Carlo

        Parameters:
        lat - struct lattice
        p - parameters for running MC
        corr_check - whether to calculate correlation function
                     or not (true or false)
    =#
```

```
m = []       #to store magnetization
m2 = []      #to store (magnetization)^2
e = []       #to store energy
e2 = []      #to store (energy)^2

if corr_check
    corr = zeros(Int64(floor(lat.L/2))+1)
end

## Start Monte Carlo
for i in 1:p.N_sweeps
    for _ in 1:lat.L^2
        ## Choose some random spin to flip
        x, y = rand(1:lat.L,2)

        ## Change in energy due to flip
        delE = del_energy(lat,x,y)

        ## Perform Metropolis check and flip
        if delE <= 0
            lat.config[x,y] = - lat.config[x,y]
        else
            check = rand(Float64)
            if check < exp(-lat.beta*delE)
                lat.config[x,y] = - lat.config[x,y]
            end
        end
    end

    ## Check if thermalization time is over
    ## if yes, check for sampling time
    if i >= p.N_thermalize && i%p.N_skip == 0
        temp, temp2 = magnetization(lat)
        push!(m,temp)
        push!(m2,temp2)
        temp, temp2 = energy(lat)
        push!(e,temp)
        push!(e2,temp2)
        if corr_check
```

```
                corr += correlation(lat)
            end
        end

        ## Print to keep track of Monte Carlo steps
        if i%1000 == 0
            println("MC step: $i")
        end
    end

    if corr_check
        ## Returning magnetization per spin, susceptibility,
        ##specific heat and correlation function

        return  mean(abs.(m))/lat.L^2,
                lat.beta*(mean(m2) - mean(abs.(m))^2)/L^2,
                lat.beta^2*(mean(e2) - mean(e)^2)/L^2,
                corr/size(m)[1]
    else
        ## Returning magnetization per spin, susceptibility
        ## and specific heat

        return  mean(abs.(m))/lat.L^2,
                lat.beta*(mean(m2) - mean(abs.(m))^2)/L^2,
                lat.beta^2*(mean(e2) - mean(e)^2)/L^2
    end
end


function correlation(lat::lattice)
    #=
        to calculate correlation function for 2D square lattice

            Parameters:
            lat: Struct lattice

            Returns an array with ith element corresponding to
            correlation function value at g(i-1)
    =#
```

```
        corr = zeros(Int64(floor(lat.L/2))+1)
        sample = zeros(Int64(floor(lat.L/2))+1)

        for i in 1:lat.L
            for j in 1:lat.L
                l = min(Int64(floor(lat.L/2)),lat.L-j)
                for k in j:j+l
                    corr[k-j+1] += lat.config[i,j]*lat.config[i,k]
                    sample[k-j+1] += 1
                end
            end
        end

        return corr./sample

    end


    function main(N_sweeps::Int64,N_skip::Int64,N_thermalize::Int64,
                  L::Int64,init::String,T::Float64,corr_check::Bool)
        #=
            main function to perform Metropolis Monte Carlo

            Parameters:
            N_sweeps: Total number of MCS
            N_skip: Sampling time
            N_thermalize: Number of MCS to skip initially
            L: Lattice size
            init: Initial configuration to use ("uniform" or "random")
            T: Temperature
            corr_check: whether to perform correlation function or not
                        (true or false)
        =#

        p = run_parameters(N_sweeps,N_skip,N_thermalize)
        lat =  lattice(L,1/T,initialize(init,L))
        temp = run_MC(lat,p,corr_check)

        ## Return magnetization per spin, susceptibility,
        ## specific heat (and correlation function)
```

```
        return temp
    end

end
```

## A.4 main_2D_wolff.jl

```
    include("Ising_2D_metropolis.jl")
    using .Ising_2D_metropolis

    N_sweeps = 5000
    N_skip = 100
    N_thermalize = 500
    L = 100
    init = "uniform"
    T = 2.0
    corr_check = false

    m, x, cv = Ising_2D_metropolis.main(N_sweeps, N_skip, N_thermalize,
                                        L, init, T, corr_check)
```

## A.5 Ising_3D_metropolis.jl

```
module Ising_3D_metropolis

    using Random
    using StatsBase

    struct run_parameters
        ## Paramters for running Monte Carlo
        N_sweeps::Int64     #Total MCS
        N_skip::Int64       #Sampling time
        N_thermalize::Int64 #Steps to discard initially
    end

    mutable struct lattice
        ## Lattice parameters
        L::Int64            #cibe size
```

```
    beta::Float64       #1/(kB*T)
    config              #storing -1/+1 for cube sites
end


function initialize(init::String,L::Int)
    #=
        Initial spin configuration

        Parameter:
        lattice - struct (ising_pars)

        Return:
        config - initial lattice configuration
    =#

    if init == "uniform"
        config = ones(L,L,L)
    elseif init == "random"
        config = 2*bitrand((L,L,L)) - ones(L,L,L)
    end

    return config
end


function periodic_boundary(index::Int64,L::Int64)
    #=
        Check for boundary index

        Parameter:
        index - Int (index to be checked for boundary condition)

        Return:
        integer
    =#

    if index == 0
        return L
    elseif index == L + 1
```

```
            return 1
        else
            return index
        end
    end


    function del_energy(lat::lattice,x::Int64,y::Int64,z::Int64)
        #=
            Calculate the change in energy after spin flip

            Parameters:
            lat - struct lattice
            x, y - Int (indices of the spin flip)

            Return:
            e - Float  (change in energy)
        =#

        e = begin 2*lat.config[x,y,z]*
            (lat.config[periodic_boundary(x-1,lat.L),y,z] +
             lat.config[periodic_boundary(x+1,lat.L),y,z] +
             lat.config[x,periodic_boundary(y-1,lat.L),z] +
             lat.config[x,periodic_boundary(y+1,lat.L),z] +
             lat.config[x,y,periodic_boundary(z-1,lat.L)] +
             lat.config[x,y,periodic_boundary(z+1,lat.L)]) end

        return e
    end


    function magnetization(lat::lattice)
        #=
            To calculate magnetizaton (order parameter)
        =#
        M = sum(sum(sum(lat.config)))
        return M, M^2
    end
```

```
function energy(lat::lattice)
    #=
        To calculate energy
    =#
    E = 0
    for i in 1:lat.L
        for j in 1:lat.L
            for k in 1:lat.L
                temp = -0.5*del_energy(lat,i,j,k)
                E += temp
            end
        end
    end
    E /=2
    return E, E^2
end


function run_MC(lat::lattice,p::run_parameters)
    #=
        To run Monte Carlo

        Parameters:
        lat - struct lattice
        p - parameters for running MC
    =#

    m = []      #to store magnetization
    m2 = []     #to store (magnetization)^2
    e = []      #to store energy
    e2 = []     #to store (energy)^2

    ## Start Monte Carlo
    for i in 1:p.N_sweeps
        for _ in 1:lat.L^3
            ## Choose some random spin to flip
            x, y, z = rand(1:lat.L,3)

            ## Change in energy due to flip
            delE = del_energy(lat,x,y,z)
```

```
                ## Perform Metropolis check and flip
                if delE <= 0
                    lat.config[x,y,z] = - lat.config[x,y,z]
                else
                    check = rand(Float64)
                    if check < exp(-lat.beta*delE)
                        lat.config[x,y,z] = - lat.config[x,y,z]
                    end
                end
            end

            ## Check if thermalization time is over
            ## if yes, check for sampling time
            if i > p.N_thermalize && i%p.N_skip == 0
                temp, temp2 = magnetization(lat)
                push!(m,temp)
                push!(m2,temp2)
                temp, temp2 = energy(lat)
                push!(e,temp)
                push!(e2,temp2)
            end

            ## Print to keep track of Monte Carlo steps
            if i%1000 == 0
                println("MC step: $i")
            end
        end

        ## Returning magnetization per spin, susceptibility
        ## and specific heat
        return  mean(abs.(m))/lat.L^3,
                lat.beta*(mean(m2) - mean(abs.(m))^2)/L^3,
                lat.beta^2*(mean(e2) - mean(e)^2)/L^3
    end


    function main(N_sweeps::Int64,N_skip::Int64,N_thermalize::Int64,
                  L::Int64,init::String,T::Float64)
        #=
```

```
            main function to perform Metropolis Monte Carlo

            Parameters:
            N_sweeps: Total number of MCS
            N_skip: Sampling time
            N_thermalize: Number of MCS to skip initially
            L: Lattice size
            init: Initial configuration to use ("uniform" or "random")
            T: Temperature
        =#

        p = run_parameters(N_sweeps,N_skip,N_thermalize)
        lat =  lattice(L,1/T,initialize(init,L))
        temp = run_MC(lat,p)

        ## Return magnetization per spin, susceptibility
        ## and specific heat
        return temp
    end
end
```

## A.6 main_3D_metropolis.jl

```
include("Ising_2D_metropolis.jl")
using .Ising_2D_metropolis

N_sweeps = 10000
N_skip = 100
N_thermalize = 1000
L = 10
init = "uniform"
T = 4.0

m, x, cv = Ising_2D_metropolis.main(N_sweeps, N_skip,
                                    N_thermalize, L, init, T)
```