Name: Adarsh Vishwakarma

1 Introduction

1.1 The Ising Model

The system consists of spins interacting through nearest neighbours. The Hamiltonian for the system is

$$H = -J \sum_{<ij>} S_i S_j, \tag{1}$$

where the summation is over nearest neighbours.
For 1D Ising model there is no phase transition whereas in two dimension the system exhibits phase transition.

1.2 Observables

By studying various observables, their variation with temperature and system sizes the phase transition of the system can be studied. The critical temperature and various critical exponents can then be determined.

1.2.1 Magnetization

The average magnetization per spin is defined as

$$\langle m \rangle = \frac{1}{N} \sum_i S_i. \tag{2}$$

1.2.2 Susceptibility

The susceptibility for the Ising model is

$$\chi = \frac{\partial \langle m \rangle}{\partial h}. \tag{3}$$

It is related to the fluctuation of the magnetization or the correlation function as,

$$\chi = \frac{1}{T} \left(\langle \delta m \rangle^2 \right) = \frac{1}{T} \left(\langle m^2 \rangle - \langle m \rangle^2 \right). \tag{4}$$

1.2.3 Binder Cumulant

It is given by

$$U_L = 1 - \frac{\langle m^4 \rangle}{3 \langle m^2 \rangle^2}. \tag{5}$$

1.3 Scaling Relations

The finite-size scaling relations for the Ising model are

$$T_c(L) = T_c + aL^{-1/\nu}, \tag{6}$$
$$\chi(L,T) = L^{\gamma/\nu} \bar{\chi}((T - T_c)L^{1/\nu}), \tag{7}$$

where

- a is a constant
- ν is the critical exponent associated with the correlation length,
- γ is the critical exponent associated with the susceptibility,
- $\bar{\chi}$ is a scaling function that is universal.

2 Numerical Solution - The Metropolis Algorithm

Consider an $n \times n$ matrix for the lattice with each entry either +1 or -1 for spins on each lattice sites. Assume that we are starting from a temperature well above critical temperature. The avgerage magnetization per spin will be close to zero.

1. The lattice is initialized with a configuration by randomly assigning +1 and -1 spin values.
2. A spin is selected randomly and the energy cost, Δ for flipping it is calculated.
3. If the energy change is negative, the flipped state is energetically favourable and the spin is flipped. If the energy cost is positive, a random number is generated in the interval [0,1]. If the generated random number is less than $exp(-\Delta/T)$, the spin is flipped otherwise it is left unflipped.
4. The step 3 is repeated N times, each time selecting spin randomly. The $N = n \times n$ such steps make up one Monte-Carlo step or MC step.

Further steps are divided into two parts: Thermalization of the lattice and Averaging the observables.

1. At a particular temperature, T, N_{th} MC steps are repeated to let the system thermalize at the temperature T.
2. Once the system is thermalized, the observables are calculated from the thermalized lattice configuration.
3. The observables are averaged over N_{av} MC steps, i.e. N_{av} number of microstates, to obtain thermodynamic values of the observables.

N_{th} can be estimated by thermalizing the lattice at low temperature. Starting from the random assignment corresponding to zero magnetization, the system will approach configuration with average magnetization one. The number of MC steps required can be taken as N_{th}. N_{av} is chosen sufficiently large to produce a nearly smooth curve variation with temperature.

3 Results

Thermalization :

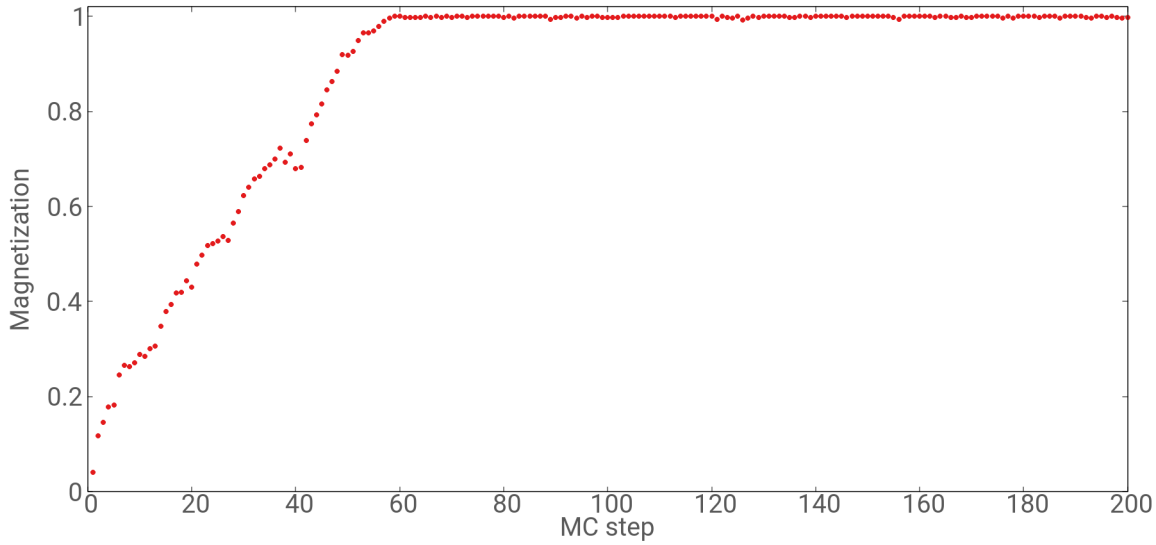

Figure 1: Thermalization of the lattice($L = 32$) at $T = 1$.

Magnetization vs Temperature :

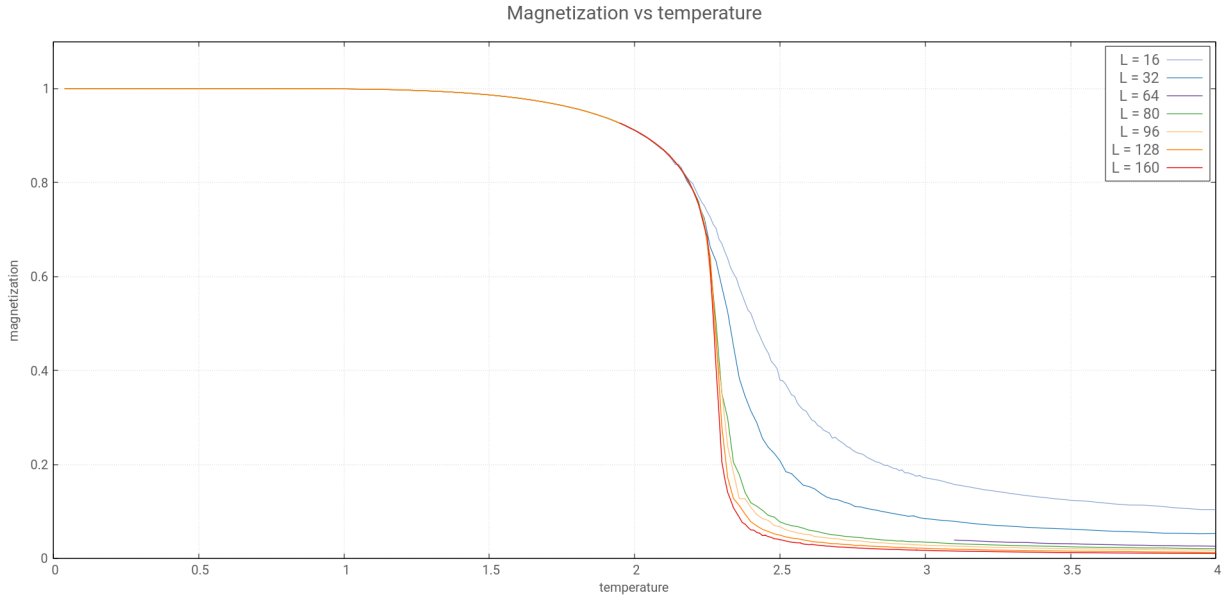

Figure 2: $m - T$ curve plotted for different system sizes.

Susceptibility vs Temperature :

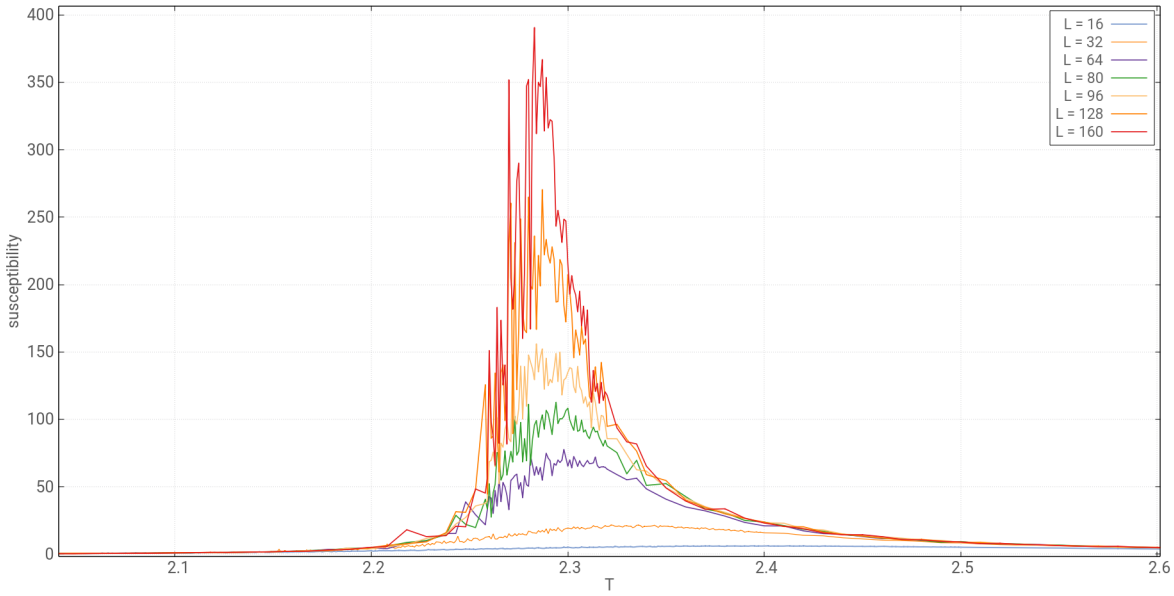

Figure 3: $\chi - T$ curve plotted for different system sizes.

Binder Cumulant vs Temperature :

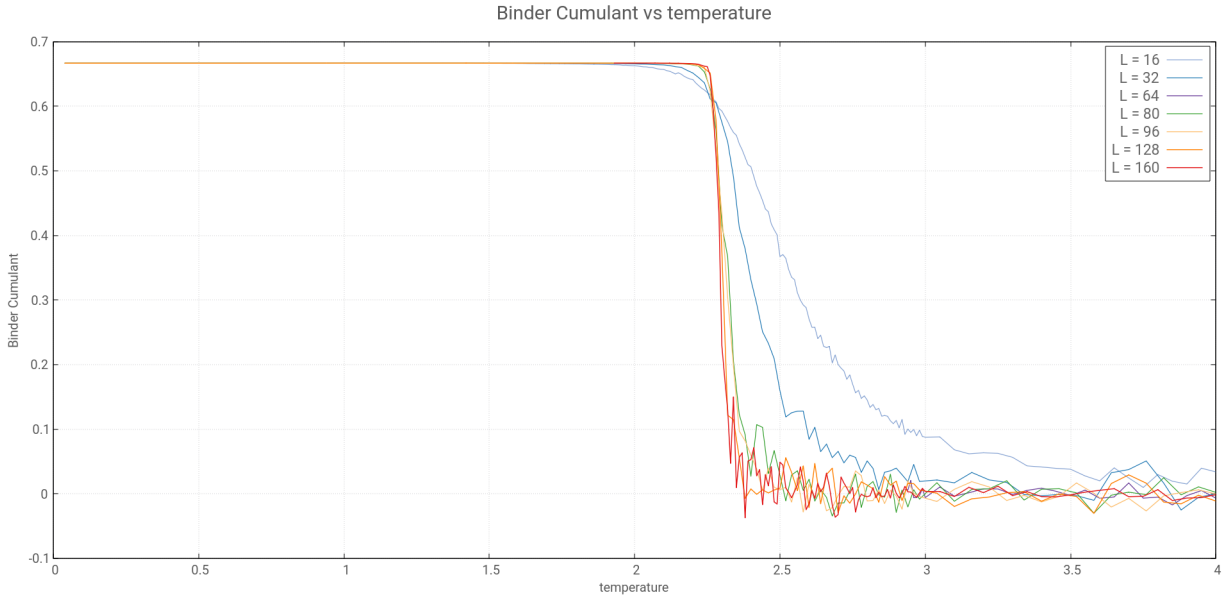

Figure 4: $B - T$ curve plotted for different system sizes.

Specific Heat vs Temperature :

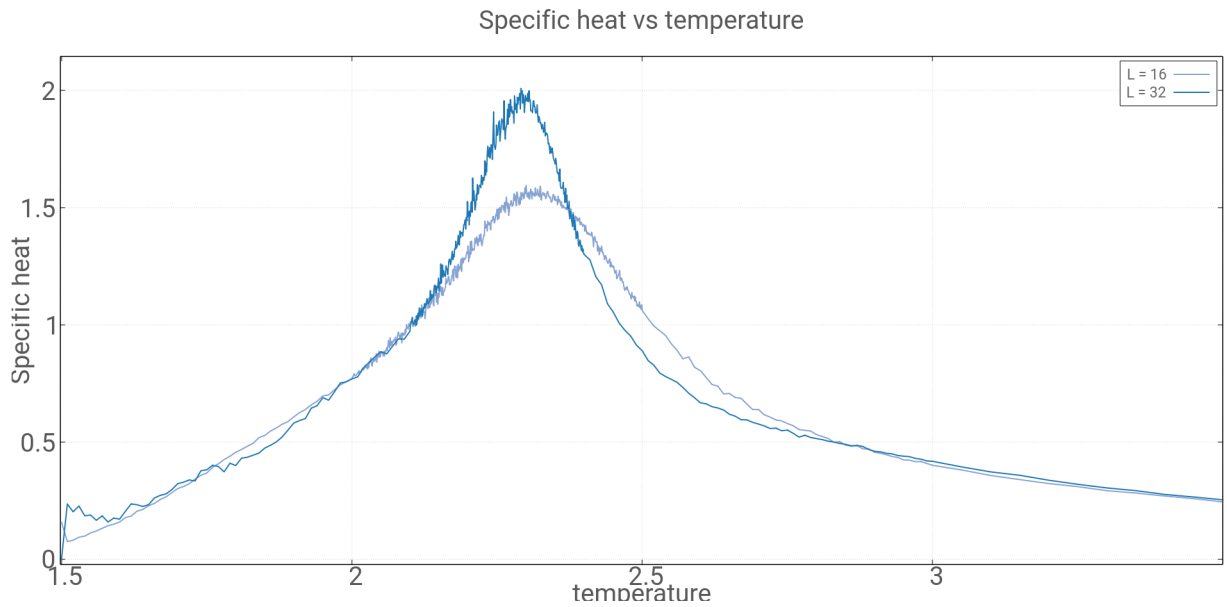

Figure 5: $C - T$ curve plotted for different system sizes.

Spin Configuration :

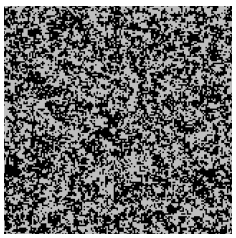

Figure 6: Spin configuration at high temperature. Lattice size 160×160

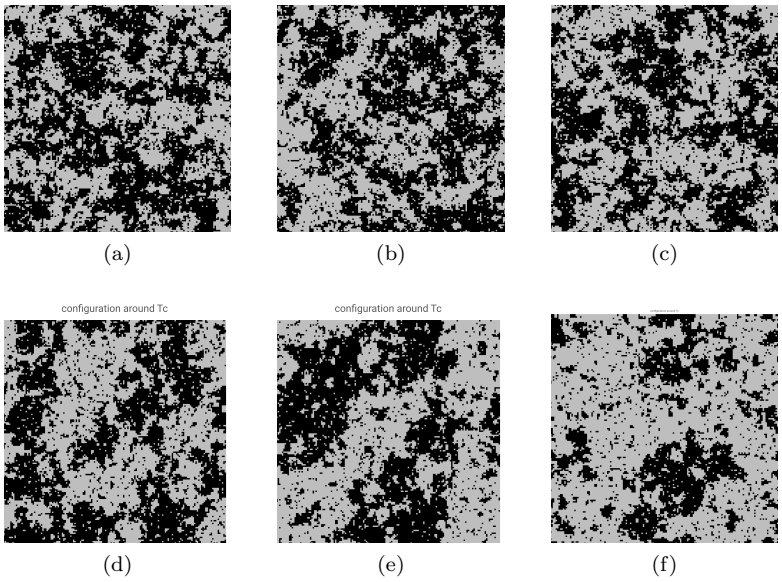

Figure 7: Spin configuration around T_c. Formation of Domains is apparent. Lattice size is 160×160

Critical Temperature :

The critical temperature is the temperature at which binder cumulant curves for different system sizes intersect. From the plot fig.(4), the intersection point is determined to be 2.27. So, the critical temperature is 2.27.

Scaling relations and critical exponents :

$T_c - L$:

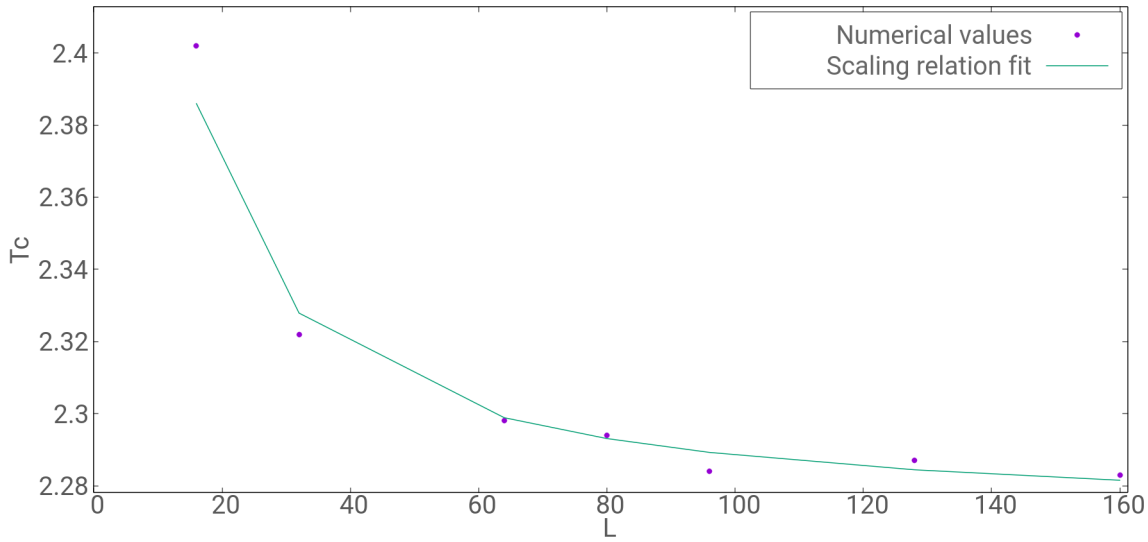

Figure 8: Scaling of critical temperature with system size.

For a given system size, T_c is determined by the peak of the susceptibility.
Fig(8) shows the finite-size critical temperature with the system size. Theoretically, the two quantities follow scaling relation 6.

Fitting the data with the scaling relation 6, the parameters and critical exponent are estimated to be

a	ν
1.870	0.997

$\chi - L$:

At $T = T_c$, using eq(7),

$$\chi(L, T_c) = pL^{\gamma/\nu}, \tag{8}$$

where p is a constant.

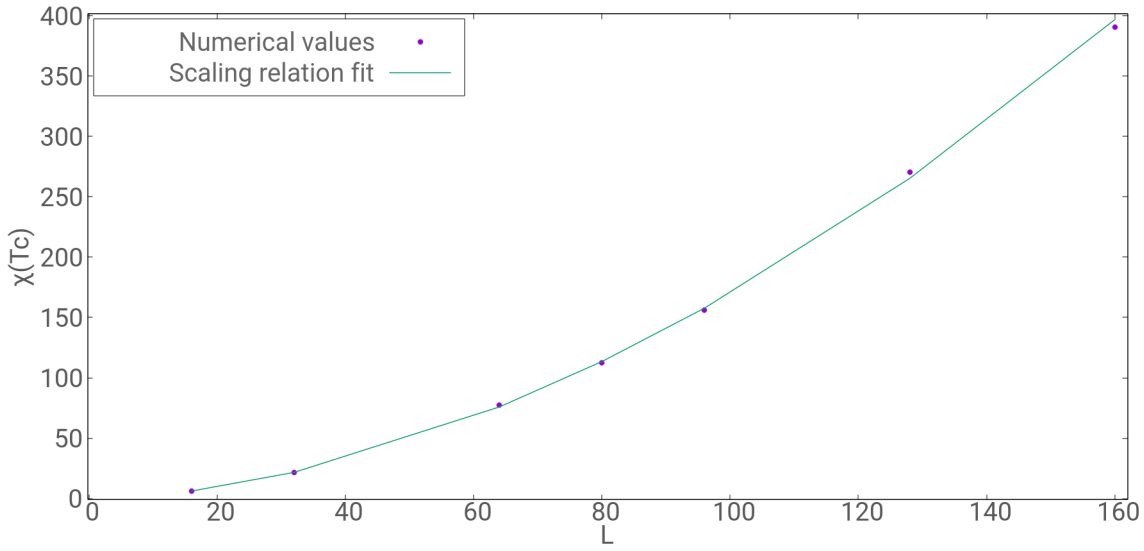

Figure 9: Scaling of susceptibility with system size.

The above figure shows the numerical values of susceptibility peak at critical temperature. The data is fitted to the scaling relation 8. The parameters are obtained below

p	γ
0.042	1.801

4 Conclusions

1. Fig 1 is a check that the Metropolis algorithm thermalizes the lattice. Starting from a random configuration at temperature 1, which is below critical temperature, successive MC steps take the system to ordered state, which is the expected behaviour.

2. At high temperature the system is in random spin configuration (fig 2). As the temperature is lowered the system transitions to an ordered state, showing that in 2D ising model there is a non-zero temperature phase transition. From fig 2, the transition seems to be more and more steep as the system size increases but the curvatures are not taking sharper form, indicating that the transition is second order.

3. The susceptibility plots in Fig(3) indicates a diverging behaviour near a certain temperature that becomes closer to $T = 2.27$ as the system size is increased. The peak for $L = 160$ occurs at $T = 2.283$. This diverging behaviour indicates second order transition of 2D ising model.

4. We see that as expected the Binder cumulant for different system sizes passes through the temperature at which phase transition occur. The intersection point renders the critical temperature $T = 2.27$.

5. Figs(6) and (7) show the spin configuration for lattice size 160. We see that at high temperature the spins are random, however, as the temperature is lowered, there start forming domains of spins around critical temperature. The size of the domains increases from (a) to (f) in the direction of decreasing temperature.

6. The critical temperature and susceptibility peak are found to follow the scaling relation. Using the finite-size scaling relation for critical temperature, the value of the critical exponent ν is estimated to be 0.997. The exact value for the 2D ising model is $\nu = 1$. The estimated value matches with the exact value within 0.3%.

7. The exact value of critical exponent γ is 1.75. The numerical estimation gives 1.801, which matches with the exact value within 2.91%.

5 Code

The code below contained in in the header file `lattice.h` creates and handles matrix for spins. It also defines a method `EnergyChangeAtFlippingSite(int xdim, int ydim)` that calculates the cost of flipping spin at the site passed to it.

```
#include<fstream>
#include<vector>
#include<random>
#include<algorithm>

using namespace std;

random_device dev;
mt19937 rng(dev());
uniform_real_distribution<> prob(0, 1);

//Prescription for 2d to 1d index conversion
//(x, y) --> y*stride + x

struct param
   {fstream mTout;
    fstream thermalization_out;
    fstream lattice_config_out;
    float temp_start = 4;
    float temp_end = 0;
    float temp_step = 0.1;
    };

class Lattice
     {public:
      vector<int> lattice, shuffled_lattice;
      int xcount, ycount, N;
      int stride, MC_step;

      double temperature;
      float lattice_energy;

      Lattice(int xdim, int ydim)
         {xcount = xdim;
          ycount = ydim;
          N = xdim*ydim;
          stride = xdim;
          MC_step = N;

          lattice.resize(N);
          shuffled_lattice.resize(N);
          iota(shuffled_lattice.begin(), shuffled_lattice.end(), 0);
          }

      Lattice()
         {xcount = ycount = 0;
          N = stride = MC_step = 0;
          }

      void ResizeLattice(int xdim, int ydim);
      void RandomAssignment();
      void OrderedAssignment();
      void OrderedAssignment(int s);
      void GetNeighbours(int x, int y, int* neighbours);
      void WriteLattice(fstream &fout);
      int EnergyChangeAtFlippingSite(int x, int y);
      };

void Lattice::ResizeLattice(int xdim, int ydim)
    {xcount = xdim;
     ycount = ydim;
     N = xdim*ydim;
     stride = xdim;
     MC_step = N;

     lattice.resize(N);
     shuffled_lattice.resize(N);
     iota(shuffled_lattice.begin(), shuffled_lattice.end(), 0);
```

```
     }

void Lattice::RandomAssignment()
    {for(int i=0; i<N; ++i)
        {lattice[i] = prob(rng)>0.5 ? 1 : -1;
         }
     }

void Lattice::OrderedAssignment()
    {fill(lattice.begin(), lattice.end(), 1);
     }

void Lattice::OrderedAssignment(int s)
    {fill(lattice.begin(), lattice.end(), s);
     }

//spin over boundaries
//boundary 1: y=0, x=0 to N-1
//boundary 2: x=N-1, y=0 to N-1
//boundary 3: y=N-1, x=0 to N-1
//boundary 4: x=0, y=0 to N-1
void Lattice::GetNeighbours(int x, int y, int* neighbours)
    {neighbours[0] = x + (y-1)*stride;
     neighbours[1] = x+1 + y*stride;
     neighbours[2] = x + (y+1)*stride;
     neighbours[3] = x-1 + y*stride;

     //apply periodic boundary condition
     if(y==0)
        neighbours[0] = (ycount-1)*stride + x;
     if(x==xcount-1)
        neighbours[1] = y*stride;
     if(y==ycount-1)
        neighbours[2] = x;
     if(x==0)
        neighbours[3] = y*stride + xcount-1;
     }

void Lattice::WriteLattice(fstream &fout)
    {for(int i=0; i<ycount; ++i)
        {for(int j=0; j<xcount; ++j)
            {fout<<j<<"\t"<<i<<"\t"<<lattice[i*stride + j]<<endl;
             }
         }
     }

int Lattice::EnergyChangeAtFlippingSite(int x, int y)
   {int neighbours[4];
    int neighbour_spin_sum=0;

    //get neighbours
    GetNeighbours(x, y, neighbours);

    for(int i=0; i<4; ++i)
       {neighbour_spin_sum += lattice[neighbours[i]];
        }

    int spin_at_xy = lattice[x + y*stride];
    //int flipped_spin_at_xy = -spin_at_xy;

    //return -(flipped_spin_at_xy - spin_at_xy)*neighbour_spin_sum;
    return 2*spin_at_xy*neighbour_spin_sum;
    }
```

The below code in `metropolis.h` file uses the `lattice.h`. It performs operations on the lattice that are related to order parameter and other observables. It defines method `ThermalizeLattice()` that thermalize the lattice at a particular temperature. The function `ThermalizeLattice()` uses function `MinimizatioForOneMCStep()` to perform one MC operation. The function `WriteMomentsVsTemperature()` writes moments of magnetization, energy, and other quantities(Susceptibility, Specific heat, Binder Cumulant) calculated from them.

```
#include"lattice.h"

using namespace std;

class IsingModel: public Lattice
     {public:
      long double magnetization, m1, m2, m3, m4, E2;
      long double variance, susceptibility, binder_cumulant, specific_heat;
      random_device rd;

      IsingModel(): Lattice()
         {magnetization = 0;
          }

      IsingModel(int xdim, int ydim): Lattice(xdim, ydim)
         {magnetization = 0;
          }

      void UpdateLatticeEnergyAndMagnetization();
      void MinimizationForOneMCStep();
      void ThermalizeLattice(long double fluctuation, int MC_in_block, int comparison_attempt);
      void ThermalizeLattice(fstream &fout, long double fluctuation, int MC_in_block, int
    comparison_attempt);
      void WriteMomentsVsTemperature(param &parameters);
      };

void IsingModel::UpdateLatticeEnergyAndMagnetization()
     {int neighbour_spin_sum, neighbours[4];

      lattice_energy = 0;
      magnetization = 0;

      for(int y = 0; y < ycount; ++y)
         {for(int x=0; x < xcount; ++x)
             {GetNeighbours(x, y, neighbours);

              neighbour_spin_sum = lattice[neighbours[0]] + lattice[neighbours[1]] + lattice[neighbours
    [2]] + lattice[neighbours[3]];

              lattice_energy += (-lattice[x + y*stride]) * neighbour_spin_sum;
              magnetization += lattice[x + y*stride];
              }
          }

      lattice_energy = lattice_energy/(2*N);
      magnetization /= N;
      }

void IsingModel::MinimizationForOneMCStep()
    {long double deltabyT;

     shuffle(shuffled_lattice.begin(), shuffled_lattice.end(), rd);

     for(int i=0; i<N; ++i)
        {int y = shuffled_lattice[i]/stride;
         int x = shuffled_lattice[i] - y*stride;

         deltabyT = (long double)(EnergyChangeAtFlippingSite(x, y))/temperature;

         if(deltabyT <= 0 || (deltabyT > 0 && prob(rng) < exp(-deltabyT)))
           {lattice[stride*y + x] = -lattice[stride*y + x];
            }
         }
      }

//Thermalize lattice
//Tries to thermalize until change in magnetization does not exceed "fluctuation"
```

```
66 //after "MC_in_block" MC steps or "comparison_attempt" is reached
67 void IsingModel::ThermalizeLattice(long double fluctuation, int MC_in_block, int comparison_attempt)
68      {long double mag1MC = magnetization;
69       long double avg_mag, avg_mag2;
70       int avg_count = 100;
71
72       start:
73       for(int i=1; i<=MC_in_block; ++i)
74          {MinimizationForOneMCStep();
75           UpdateLatticeEnergyAndMagnetization();
76           }
77
78       //calculate average magnetization using avg_count points
79       avg_mag = avg_mag2 = 0;
80       for(int i=1; i<=avg_count; ++i)
81          {avg_mag += magnetization;
82           avg_mag2 += magnetization*magnetization;
83
84           MinimizationForOneMCStep();
85           UpdateLatticeEnergyAndMagnetization();
86           }
87        avg_mag /= avg_count;
88        avg_mag2 /= avg_count;
89
90        //thermalize again if fluctuation is larger than threshold
91        if(sqrt(avg_mag2 - avg_mag*avg_mag) > fluctuation && comparison_attempt--)
92          {goto start;
93           }
94
95        //extra thermalization to ensure thermalized state
96        for(int i=1; i<=15*MC_in_block; ++i)
97           {MinimizationForOneMCStep();
98            UpdateLatticeEnergyAndMagnetization();
99            }
100       }
101
102 //also writes thermalization data to the given file
103 void IsingModel::ThermalizeLattice(fstream &fout, long double fluctuation, int MC_in_block, int
      comparison_attempt)
104     {long double mag1MC = magnetization;
105      long double avg_mag, avg_mag2;
106      int avg_count = 100;
107      int MC_count = 0;
108
109      start:
110      for(int i=1; i<=MC_in_block; ++i)
111         {MinimizationForOneMCStep();
112          UpdateLatticeEnergyAndMagnetization();
113
114          fout<<++MC_count<<"\t"<<magnetization<<endl;
115          }
116
117       //calculate average magnetization using avg_count points
118       avg_mag = avg_mag2 = 0;
119       for(int i=1; i<=avg_count; ++i)
120          {avg_mag += magnetization;
121           avg_mag2 += magnetization*ma gnetization;
122
123           MinimizationForOneMCStep();
124           UpdateLatticeEnergyAndMagnetization();
125
126           fout<<++MC_count<<"\t"<<magnetization<<endl;
127           }
128        avg_mag /= avg_count;
129        avg_mag2 /= avg_count;
130
131        if(sqrt(avg_mag2 - avg_mag*avg_mag) > fluctuation && comparison_attempt--)
132          {goto start;
133           }
134
135        for(int i=1; i<=3*MC_in_block; ++i)
136           {MinimizationForOneMCStep();
137            UpdateLatticeEnergyAndMagnetization();
```

```
138
139           fout<<++MC_count<<"\t"<<magnetization<<endl;
140           }
141        }
142
143 void IsingModel::WriteMomentsVsTemperature(param &parameters)
144    {long double fluctuation = 0.01;
145     int no_of_MC_blocks = 100;
146     int MC_in_block = 10000;
147     int total_MC = no_of_MC_blocks*MC_in_block;
148
149     long double avg_magnetization, avg_abs_magt, avg_energy;
150     int comparison_attempt = 8;
151
152     //write lattice configuration before thermalization
153   //  WriteLattice(parameters.lattice_config_out);
154   //  parameters.lattice_config_out<<endl<<endl;
155
156     for(temperature = parameters.temp_start; temperature >= parameters.temp_end; temperature -=
    parameters.temp_step)
157        {avg_magnetization = avg_abs_magt = avg_energy = 0;
158         m1 = m2 = m3 = m4 = E2 = 0;
159
160         //thermalize the lattice
161         if(temperature == parameters.temp_start)
162           {ThermalizeLattice(parameters.thermalization_out, fluctuation, MC_in_block, comparison_attempt
    );
163            }
164         else
165            ThermalizeLattice(fluctuation, MC_in_block, comparison_attempt);
166            //lattice thermalized
167
168            //First four magnetization moments about zero
169            //and first two moments of lattice energy about zero
170            for(int i=0; i<total_MC; ++i)
171               {MinimizationForOneMCStep();
172                UpdateLatticeEnergyAndMagnetization();
173
174                avg_magnetization += magnetization;
175                m1 = abs(magnetization);
176                avg_abs_magt += m1;
177                m2 += (m1*m1);
178                m3 += (m1*m1*m1);
179                m4 += (m1*m1*m1*m1);
180                avg_energy += lattice_energy;
181                E2 += lattice_energy*lattice_energy;
182                }
183
184         avg_energy /= total_MC;
185         E2 /= total_MC;
186         avg_magnetization /= total_MC;
187         avg_abs_magt /= total_MC;
188         m1 = avg_abs_magt;
189         m2 /= total_MC;
190         m3 /= total_MC;
191         m4 /= total_MC;
192         variance = (m2 - m1*m1);
193         susceptibility = variance/temperature;
194         binder_cumulant = 1 - m4/(3*m2*m2);
195         specific_heat = (E2 - avg_energy*avg_energy)/(temperature*temperature);
196         //Moments calculated
197         parameters.mTout<<temperature<<"\t"<<avg_magnetization<<"\t"<<avg_abs_magt<<"\t";
198         parameters.mTout<<m2<<"\t"<<m3<<"\t"<<m4<<"\t";
199         parameters.mTout<<variance<<"\t"<<susceptibility<<"\t"<<binder_cumulant<<"\t";
200         parameters.mTout<<avg_energy<<"\t"<<E2<<"\t"<<specific_heat<<"\t"<<endl;
201
202         //write thermalized lattice configuration at temperature temp_start
203       //   WriteLattice(parameters.lattice_config_out);
204       //   parameters.lattice_config_out<<endl<<endl;
205          }
206       //write thermalized lattice configuration at temperature temp_end
207     //  WriteLattice(parameters.lattice_config_out);
208       }
```

The below code uses the header file `metropolis.h` to simulate the 2D ising model. The various moments, and other observables calculated at each temperature are stored in file `mtcurve[dim].txt`, where `dim` is the dimension of the lattice. The data for the thermalization of the lattice is stored in the file `thermalization[dim].txt`. The configuration of the lattice at different temperatures are stored in the file `latice_config[dim].txt`.

```
#include<iostream>
#include<time.h>
#include"metropolis.h"

using namespace std;

int main()
   {clock_t start, end;
    start = clock();

    int dims[] = {2, 4, 8, 16, 32, 64};
    param parameters[7];
    IsingModel model[7];

    for(int i=0; i<=0; ++i)
       {//Create lattice of the desired dimension
        model[i].ResizeLattice(dims[i], dims[i]);

        //create various files for the data
        //file for moments and other observables
        parameters[i].mTout.open("mtcurve" + to_string(dims[i]) + ".txt", ios::out|ios::trunc);
        //file for thermalization of the lattice with MC steps
        parameters[i].thermalization_out.open("thermalization" + to_string(dims[i]) + ".txt", ios::out|
    ios::trunc);
        //file for the configuration of the lattice with temperature
        parameters[i].lattice_config_out.open("lattice_config" + to_string(dims[i]) + ".txt", ios::out|
    ios::trunc);

        //set initial temperature
        model[i].temperature = 4;
        //assign a random configuration
        model[i].RandomAssignment();
        //update the lattice for its energy and magnetizxation at the present configuration
        model[i].UpdateLatticeEnergyAndMagnetization();

        parameters[i].mTout<<"Temperature\t <m>\t <|m|>\t <m^2>\t <|m|^3>\t <m^4>\t";
        parameters[i].mTout<<"Variance\t Susceptibility\t Binder Cumulant"<<"\t";
        parameters[i].mTout<<"<Energy>\t <Energy^2>\t Specific Heat\t"<<endl;

        //write the various quantities in the files present in the parameters structure
        model[i].WriteMomentsVsTemperature(parameters[i]);
        }

    end = clock();
    cout<<"\nRun time: "<<double(end-start)/double(CLOCKS_PER_SEC)<<endl;

    return 0;
    }
```