Monte Carlo Simulations of Ferromagnetism via the 2D Ising Model

1 Introduction

Monte-Carlo methods represent a class of computational methods that utilize random numbers to model inherently stochastic processes or perform calculations where you can leverage the properties of large numbers of random variables. In the context of statistical mechanics, Markov Chain Monte-Carlo methods provide a means to simulate model systems and extract their equilibrium properties without needing to keep track of the dynamical details of their composite particles. Here I utilize Monte-Carlo techniques to study the behavior of ferromagnetic materials under the 2D Ising model. Ferromagnets (named after iron, Fe), are materials in which finite size domains of spontaneous magnetization form below some critical temperature.

The Ising model in the context of magnetism describes how a collection of two-spin-state lattice sites connected to a heat bath evolves and interacts. The Ising model is easily solved analytically in 1D, tractable, yet much more involved in 2D (requiring elliptic integrals), and has never been solved in the 3D case. It is therefore to our advantage to take a numerical approach towards understanding these systems. A popular approach to computing the 2D Ising model is the Metropolis algorithm,[32] which simulates the behavior of a large collection of spins in a way that follows Boltzmann statistics, by treating lower energy configurations preferentially but allows the possibility of higher energy configurations.

As a part of this project, I have implemented the Metropolis algorithm for the 2D Ising model at zero applied field. With this, I demonstrate domain formation and thermal equilibriation with a time series of lattice configurations. From this, I will then on to calculate the average energy, magnetization, and specific heat of the system and their temperature dependencies. This allows for quantitative treatment of identifying the ferromagnetic phase transition's critical temperature.

2 Theory

Ferromagnetic materials are materials for which below some critical temperature T_C, the spins on each atom spontaneously align in the same direction as one another over some finite region, referred to as a domain [1]. While ferromagnetism is an inherently quantum phenomenon, a surprising amount of insight can be extracted using thermodynamic considerations alone. The first person who was successful in approaching this problem was Ernst Ising, a student of Wilhelm Lenz, who analytically solved a simple 1D lattice model for magnetism. His approach, which became known as the Ising model, was furthered by Lars Onsager who found an exceedingly difficult analytic solution to the 2D Ising model, and no one to this day has found a solution to the corresponding 3D Ising model.

Fortunately for us, we have computers at our disposal that can generate large quantities of random numbers which can be used in a very simple manner to simulate the stochastic evolution of a spin lattice system in two or three dimensions. This type of approach is known as a Markov Chain Monte Carlo (MCMC) method. The Mathematical formalism of Markov chains can be used to analyze which sorts of discrete stochastic models can be guaranteed to converge in a way that agrees with statistical thermodynamics. The conditions on any viable model are that the transition

Figure 1: Optical image of a ferromagnetic domain in a silicon steel obtained by Kerr microscopy.

probabilities between states depend only on the current state of the system (no memory). That every possible configuration must be reachable with nonzero probability (ergodicity), and that the model satisfy the principle of microscopic reversibility (detailed balance).[5] There are many such suitable algorithms for solving the Ising model such as the Heat Bath, Wolff, Metropolis, and Wing-Landau Sampling algorithms[52],[3] but for our purposes we will only consider the Metropolis algorithm.

The Hamiltonian for the Ising Model is given as

$$\mathcal{H} = -\sum_{<i,j>} J s_i s_j - B\mu_b \sum_{<i>} s_i \tag{1}$$

Here, s_i is the spin state of a given site of the lattice. The first term is a sum over each lattice site's nearest neighbors and accounts for spin-spin exchange interactions. In general, the exchange interaction parameter, J, can be positive or negative, corresponding to ferromagnetic or anti-ferromagnetic (spins tend to anti-align below T_C) behavior, respectively. The second term accounts for individual spins interactions with an external magnetic field, B.

With this expression for the energy of a given lattice configuration and the Boltzmann probability distribution

$$P \propto exp(-E/k_B T) \tag{2}$$

we are equipped to state the steps of the Metropolis algorithm,[2]:[3]

1) Arbitrarily initialize the spin state of the lattice.

2) Pick a lattice site at random and calculate the energy of the configuration.

3) Flip its spin state and calculate the new energy of this trial configuration.

4) Accept the trial configuration if $E_{Trial} < E_{Initial}$, OR if a random number uniformly distributed [0,1] is less than $R = exp(-\Delta E/k_B T)$.

This algorithm is then applied iteratively until our system reaches thermodynamic equilibrium. A little more algebra simplifies the calculation of energies such that only one sum over

nearest neighbors has to be performed:

$$R = exp(-(E_{Trial} - E_{Initial})/k_B T) \tag{3}$$

$$E_{Trial} = -s_o(s_{+x} + s_{-x} + s_{+y} + s_{-x}) - s_o B\mu_b = -s_o(\sum +B) \tag{4}$$

$$E_{Initial} = s_o(s_{+x} + s_{-x} + s_{+y} + s_{-x}) + s_o B\mu_b = s_o(\sum +B) \tag{5}$$

$$R = exp(-2s_o(\sum +B)/k_B T) \tag{6}$$

To quantify some of the behavior of our system, we calculate 3 thermodynamic quantities: Average energy $< E >$, specific heat C, and magnetization $< M >$. In zero applied field, the energy and magnetization are simply given as

$$E = -\sum_{<i,j>} J s_i s_j \tag{7}$$

$$M = \sum_{<i>} s_i \tag{8}$$

Due to the statistical fluctuations in our model, it turns out to be best to calculate the specific heat in terms of energy fluctuations.

$$C \equiv \frac{dU}{dT} = \frac{< E^2 > - < E >^2}{N^2 k_b T^2} \tag{9}$$

where N is the size of our square lattice. By calculating the temperature dependencies of these quantities, we hope to observe results consistent with domain formation below a certain T_C.

Throughout this simulation we enforce periodic boundary conditions by defining our coordinates modulo 50 in the x direction and modulo 25 in the y direction. Not only does this save us a lot of grief in dealing with stepping out of our array bounds, but it allows us to simulate the physical environment of a very large but homogeneous space with only a relatively small local sample space. In the literature, these are refereed to as Born von Karman boundary conditions.

3 Implementation

In order to implement our Monte-Carlo method, we must have a means to generate random numbers. For this, we use the pseudo-random number generator Ranq1.doub().[4] We are free to choose what integer seed to use when we declare Ranq1. It is helpful to declare two instances, one with a constant seed for developing our code and testing it on various machines, and a dynamic seed that will produce a different random number sequence each run. For the latter case, we use time(NULL) which passes the time in seconds since Jan 1, 1970.

A single, 2D array is initialized with values -1,1 assigned at random. This represents a "hot" start from a disordered state. We just as easily could have chosen a "cold" start configuration

where all spins were aligned initially and then warmed through T_C. The results for the underlying thermodynamic quantities are largely the same.

To understand some of the features of this program, it is helpful to define a variable *sweeps* which is equal to the number of lattice points *NxN*. This lets us consider how our algorithms tend to scale in number of iterations required to equillibriate, without reference to our choice of lattice size (in this case 500x500). Throughout the simulation, we set J = k_B = 1, which amounts to measuring energies and temperatures in terms of the exchange interaction J. This simplifies our analysis and does not detract from the functional form of our thermodynamic quantities.

One of the first steps towards a quantitative treatment of the 2D Ising model is to determine how long it takes for the system to equillibriate. To do this, I plotted average energy versus time (number of MCMC iterations) and looked at the asymptotic behavior at for two representative temperatures [2]. The system at T=2.0 converges much slower than the system at T = 5.0 but has less thermal noise. From this we can deduce two things about how the temperature dependence of E,C, and M will need to be calculated. First, these averages will need to be sampled over a sizable sweep window to account for statistical fluctuations at higher T. Second, we must pick a range in which the average energy does not vary greatly (i.e. the system is in equilibrium) over the averaging window in question.

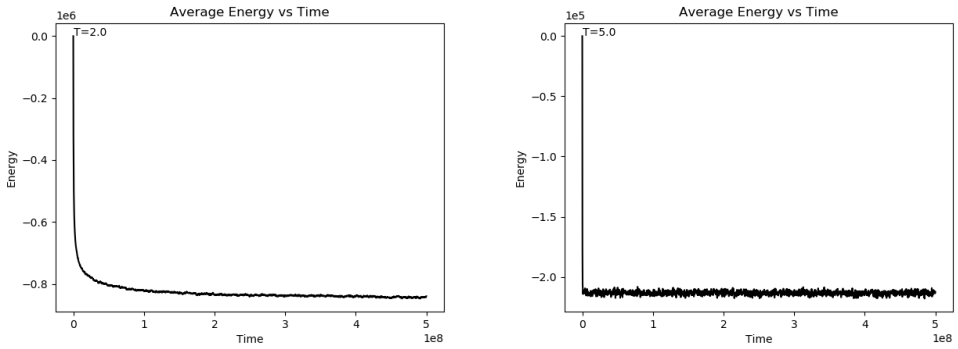

Figure 2: Plots of Average Energy vs. Time at T = 2.0,5.0 used to determine suitable equilibriation times. Note that the lower temperature model converges slower but has less thermal noise.

Playing with these parameters a bit, I found that equillibriation runs of 1000 sweeps and averaging runs of 200 sweeps produced clean, reproducible plots of E,C, and M. Interestingly enough, I also found that the first several temperature runs following the initialized lattice would produce inaccurate results, regardless of starting temperature. To fix this, I included an single long equillbriation run of 20,000 sweeps before beginning to traverse a temperature range. This successfully removed any influence of starting conditions from the temperature dependence plot.

Once this was resolved, the rest of the implementation was relatively straight forward. After initializing the lattice, you are free to select either a single temperature or a range of temperatures to study. For a single temperature, you may observe domain formation in time as a function of time by outputting the lattice configuration to a file at select numbers of iterations [3]. Alternatively, you may allow the system to equillibriate fully, output the resulting configuration, and then repeat this process for different temperatures. With this you may observe domain formation as a function of temperature [4].

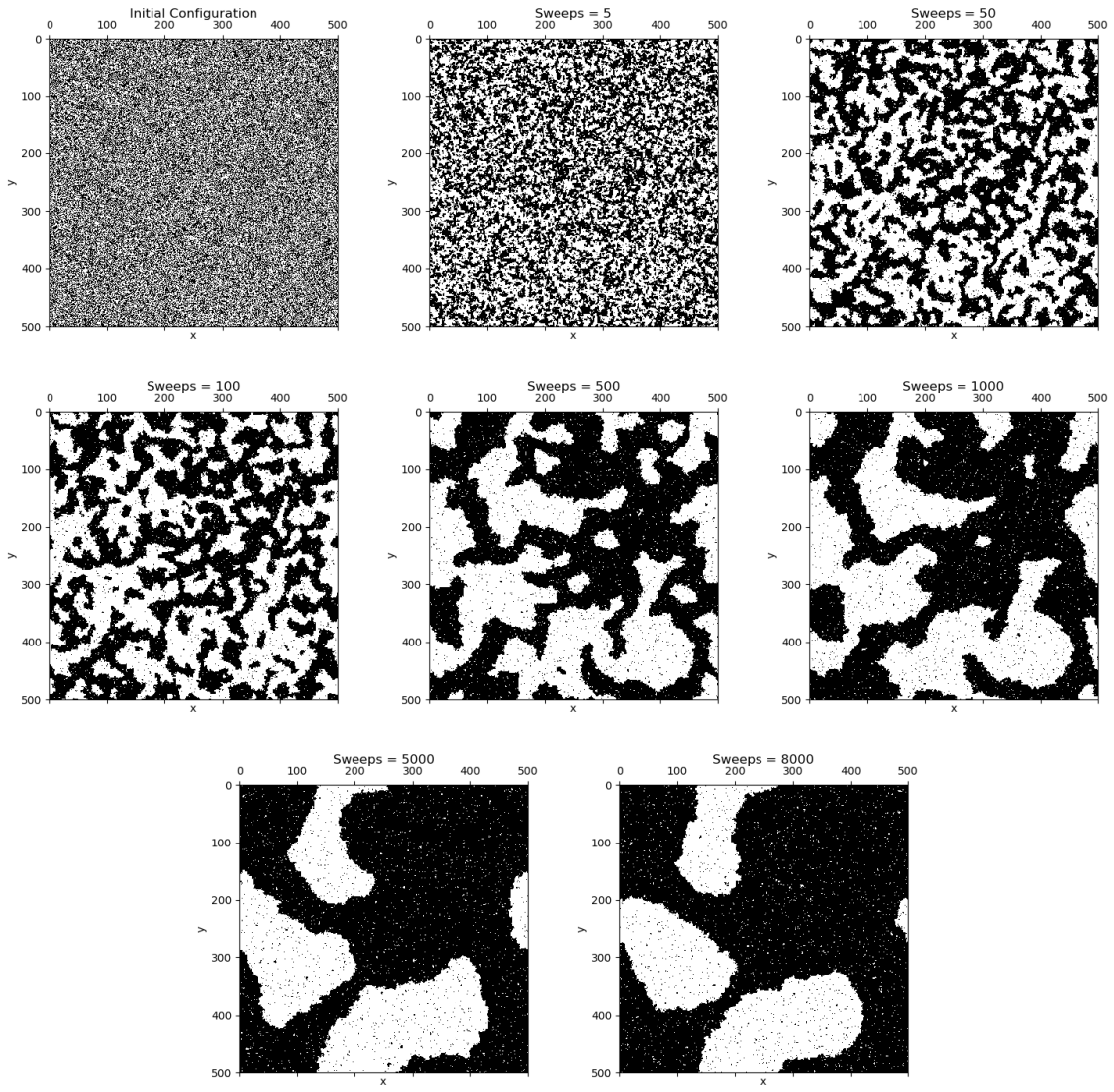

Figure 3: Time evolution of domain clusters at temperature T=1.2, arranged by number of sweeps through the lattice. Note that by 5000 sweeps, our configuration has roughly stabilized, indicating thermal equilibrium has been reached

To calculate E(T), C(T), and M(T), a temperature sweep is required. This is done simply by embedding the bulk of the body code into one large for-loop where temperature is incremented each step. For one temperature iteration the lattice is equillibriated using 1000 sweeps, then run for 200 more sweeps as $< E >$,$< E^2 >$,$< M >$, and C are calculated and output to a file. It is VERY important that you do not reinitialize the lattice with random values as a part of this temperature loop! Since the random number generator returns different random values during a run, doing this would correspond to calculating E,C, and M for a *different* configuration of spins at each temperature value, as opposed to altering the temperature of a single configuration and allowing it to equilibriate. Interestingly enough, a plot of average energy is not effected by such a faux-pas but specific heat and magnetization are (and drastically so). This makes intuitive sense

as from the definition of systems at thermal equilibrium. We are working with systems in which average energy has settled down (not energy fluctuations or spin configurations=, which C and M depend on).

4 Results and Analysis

We begin our analysis by studying a rather gratifying time series visualization of ferromagnetic domain formation [3]. This shows the case of a "hot" start where a system goes through its phase transition as its temperature equilibriates to a value lower than T_C. Not only does this give us clear indication that our model is working, but it provides a geometrical intuition for what is going on in the first plot of [2] where for a low T system a great deal of the equillibriation happens very quickly but the rest takes much longer.

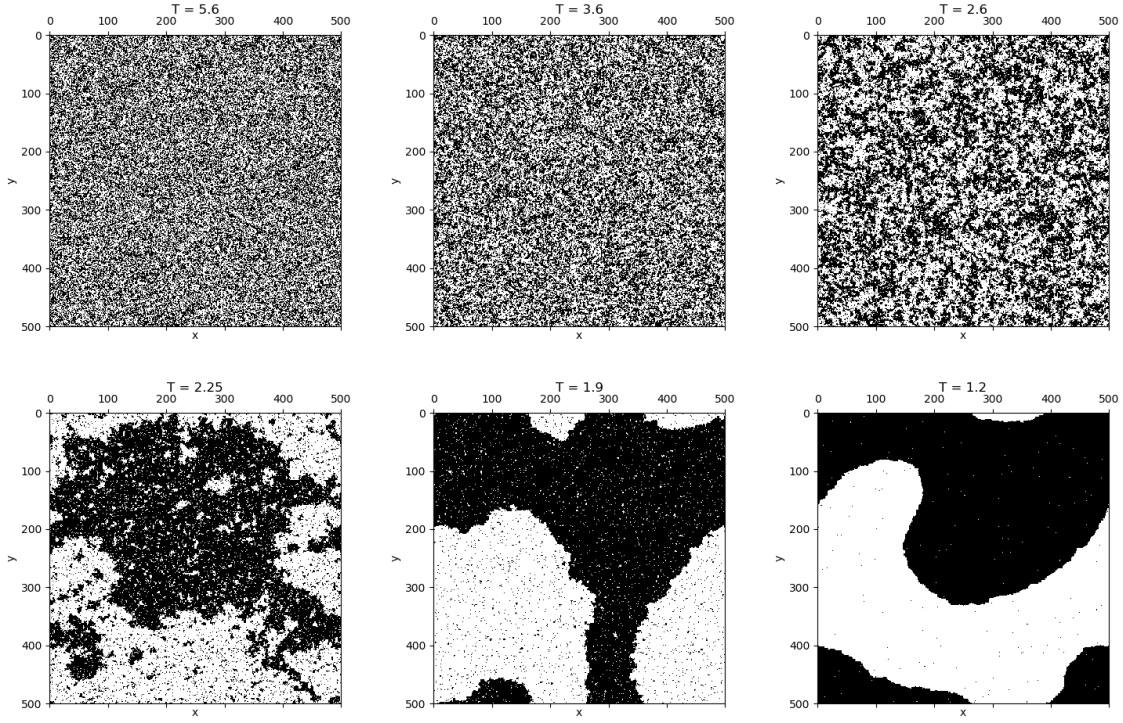

Figure 4: Plots of Domain states in thermal equilibrium at various temperature values. Note how domains of systems at lower T have less "speckles" indicating less thermal fluctuation.

Next, we consider a series of systems at equilibrium for different temperature values [4]. These differ from the previous series in that they are all equillibriated at their respective temperatures. The first three figures do not show domain formation, whereas the last two clearly do. The fourth figure is something special. It is the system equillibriated exactly at the critical temperature $T_C = 2.25$ and it displays a number of special characteristics to be discussed shortly. It is also interesting to note how that even at thermal equillibrium, these systems are still quite dynamic. It is possible to roughly determine the level of thermal fluctuations (exactly like those present in [2]) by how "speckled" the regions are. More speckling indicates spins with enough thermal energy to

flip against their respective domains.

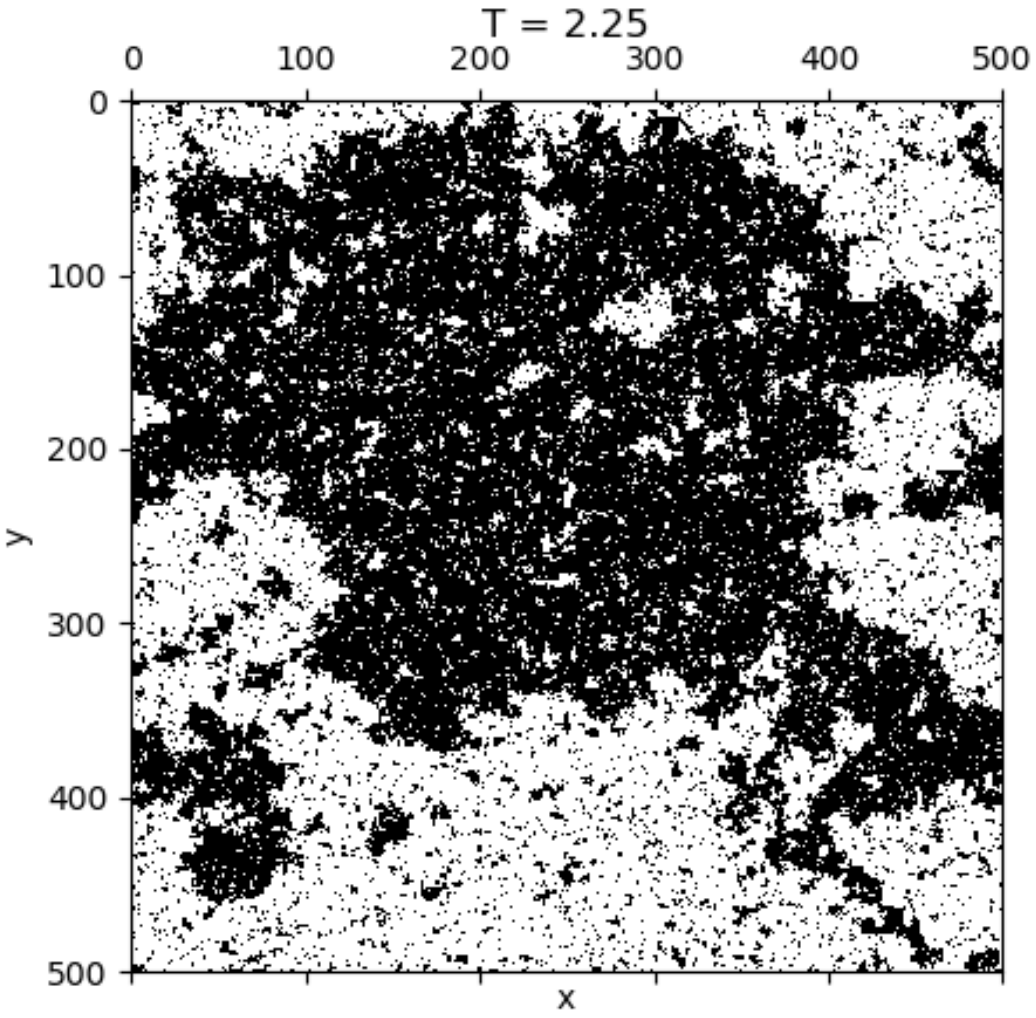

Figure 5: Configuration of our equilibrium system at T = T_C = 2.25. Note the the self-similar cluster-within-a-cluster behavior indicative of a critical phase transition.

We now examine the T = 2.25 case a little further [5]. Examining the figure closely, we notice that any small region of our system looks statistically similar to the system as a whole. This self similar fractal behavior of clusters within clusters within clusters is the first sign that we have accurately modeled a critical phase transition. If this is correct, we should also be able to observe a divergence in the specific heat of our material at said temperature. Figure [7] shows exactly that, producing nice exponential behavior on both sides of T_C.

As further confirmation, we may look at the average energy and magnetization of our material as a function of temperature. Indeed the average energy dips downward at T_C, indicating that domain formation lowers the total energy of the system as previously opposing spins start to align. Lastly, if it was not obvious from the picture, the plot of magnetization vs. temperature displays

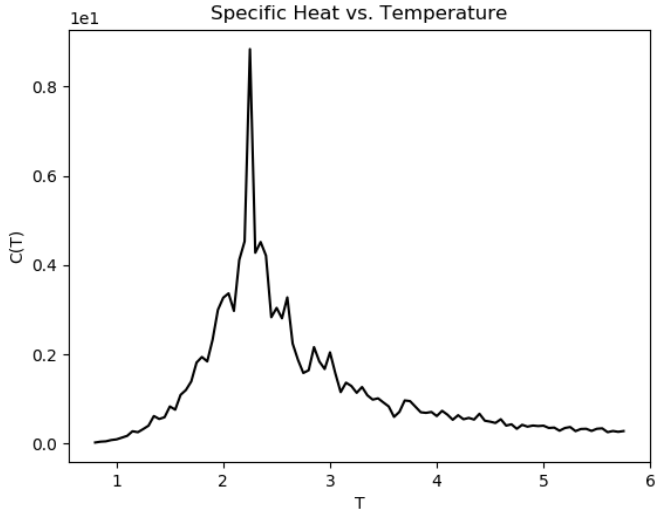

Figure 6: Plot of average energy ¡E¿ vs. temperature. Note the reduction of average energy around T_C as individual spins begin to align.

sharp step-like behavior at T_C as the material spontaneously magnetizes along one direction.

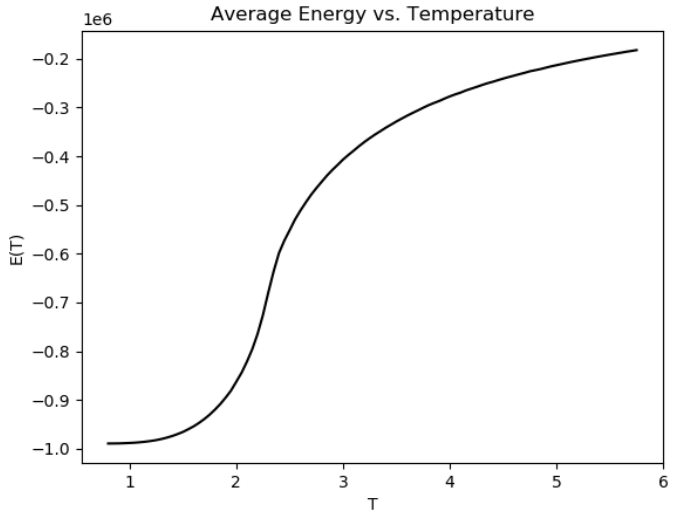

Figure 7: Plot of specific heat vs. temperature. Note the divergence at $T_C = 2.25$, indicative of a critical phase feature in the material.

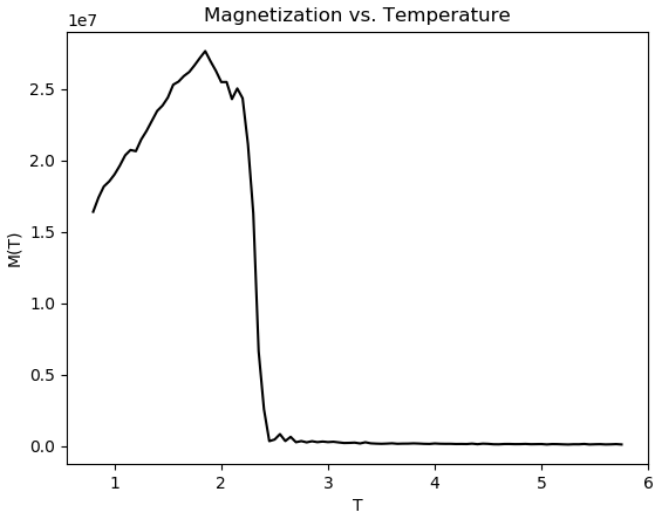

Figure 8: Plot of magnetization vs. temperature. Note the sharp step behavior at $T_C = 2.25$, indicating a spontaneous magnetization of the material.

5 Conclusion

In summary, by implementing the Metropolis algorithm - a Markov Chain Monte Carlo method - to a 2D lattice of spins with defined interaction energies, we were able to simulate the thermodynamic behavior of a ferromagnet in a way that was consistent with Boltzmann statistics. In terms of simulating real matter, this is clearly a very crude model. We completely neglected atomic and electronic structure and assumed only nearest neighbor, semiclassical spin interactions. Yet despite all this, our model does remarkably well at demonstrating one of the defining features of ferromagnetic materials - a critical phase transition! Not only did our system demonstrate the qualitative behavior of domain formation and equillibriation, but the average energy, specific heat, and magnetization displayed temperature scaling of the exact functional form one would expect from a real system.

Although this captured arguably the most important physics of ferromagnets, extensions of this model could be used to calculate even more behaviors of interest. For example Prof. Jim Sethna has done fascinating work using a 3D lattice Ising Model to study the avalanche behavior of many spin clusters flipping at once in a material and how this results in jumps in hysteresis loops and the effect known as Barkhausen noise, of when a ferromagnet is pushed through a coil of wire connected to a speaker[5].[1] Moreover, similar extensions of the Ising model and other lattice models can be used to model binary alloys, liquid-gas transitions, and quantum path integration for quantum field theories.[52]

In order to approach some of these more involved topics, performance improvements of our MCMC solver would be desirable. As outlook I propose two different approaches to boosting performance. The first would be to use an adaptive method similar to the Runge-Kutta-Fehlberg method,[4] in order to identify proper thermal equilllibriation in runs performed at different temperatures. That way, when doing a temperature sweep, you would not be as limited by the slowest

converging case. The second such performance boost could come from finding a way to run this method in parallel. If you only needed to calculate average properties, a simple solution would be to run two separate Ising models in parallel and average you results. If you were after the final configuration of a large size or 3D lattice, another approach could be to use one core as a dedicated random number generator in an SMP scheme, with the other handling the rest of the computations. With shared memory, this approach might just be viable on a personal laptop, offering more freedom in exploring fun Monte Carlo simulations.

References

[1] Karin Dahmen and James P. Sethna. Hysteresis, avalanches, and disorder-induced critical scaling: A renormalization-group approach. *Phys. Rev. B*, 53(22):14872–14905, June 1996.

[2] Rubin H. Landau, Jose Paez, and Cristian C. Bordeianu. *A Survey of Computational Physics: Introductory Computational Science*. Princeton University Press, Princeton, NJ, USA, har/cdr edition, 2008.

[3] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, June 1953.

[4] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[5] James P. Sethna. Statistical mechanics: Entropy, order parameters, and complexity. In *Oxford Master Series in Physic*. Univ. Press, 2006.

Source Code Listing

```
/* AEP 438 Final Project

Run on a core i7 running macOS Mojave with Xcode

Matt Danbury 12-Dec-2018

Explore the thermodynamic behavior of ferromagnets by using Markov Chain Monte Carlo
methods to solve the 2D Ising model.

*/

#include <cstdlib>
#include <cmath>
#include <iostream>
#include <fstream>
#include <iomanip>

#include "nr3.h"
#include "ran.h"
#include "arrayt.hpp"

using namespace std;
inline int periodic( int i, int n ) {return((i+n)%n);}
int neighbSum(arrayt<int>& Lattice, int rx, int ry);

int main(){
ofstream fp;
int i, j, k, m, h, rx, ry, ns, seed = 4321, Nsites = 500, sweep = Nsites*Nsites,
nmc = sweep*400, mMax = 200;   // seed = time(NULL)
double relProb, Mstate, Mavg, Estate, Eavg, Efluct, C , kB = 1.0, T = 1.2 ,jExch = 1.0, bField = 0.0;
arrayt<int> Lattice(Nsites,Nsites);
Ranq1 MCran(seed);

fp.open("quantities_mintest.txt");
if( fp.fail() ) {cout << "Cannot open file" << endl; exit(0);}

//Initialize spin lattice
for (i=0; i<Nsites; i++) {
for (j=0; j<Nsites; j++) {
if (0.5 > MCran.doub())
Lattice(i,j) = 1;
else
Lattice(i,j) = -1;
}}

// Temperature Sweep to Determine Thermodynamic Properties. Set h = 100 to
// observe phase transition, h=1 to pull configuration plots at a single T value.
for(h=0;h<100;h++){
cout << "h run " << h << endl;

// Equillibriation Run
for (k=0; k<nmc; k++) {
rx = floor(Nsites*MCran.doub());            //Select a lattice site at random
ry = floor(Nsites*MCran.doub());
ns = neighbSum(Lattice, rx, ry);
relProb = exp(-2.0*Lattice(rx,ry)*(jExch*ns + bField)/(kB*T));  // Calculate Boltzmann probablity value
```

```
if (MCran.doub() < relProb) Lattice(rx,ry) *= -1;           // Flip Site if thermodynamically favorable
}

// Calculations of Average Energy, Magnetization, and Specific Heat
Eavg = 0.0; Mavg = 0.0; Efluct = 0.0;

for (m=0; m< mMax*sweep; m++) {          // Time Integration Window: Nsites*Nsites*200 iterations
rx = floor(Nsites*MCran.doub());            // has worked well for 500x500 maps.
ry = floor(Nsites*MCran.doub());
ns = neighbSum(Lattice, rx, ry);
relProb = exp(-2.0*Lattice(rx,ry)*(jExch*ns + bField)/(kB*T));
if (MCran.doub() < relProb) Lattice(rx,ry) *= -1;

if (m % sweep == 0){              // At regular sampling intervals, pull Energy and Magnetization from
Estate = 0.0; Mstate = 0.0;       // lattice
for (i=0; i<Nsites-1; i++) {
for (j=0; j<Nsites-1; j++) {
Estate += (-jExch * neighbSum(Lattice, i, j) * Lattice(i,j));
Mstate += Lattice(i,j);
}}

Eavg += Estate / mMax;                      // Average Energy, Fluctionations, and Magnetization over
Efluct += (Estate * Estate) / mMax;         // time
Mavg += abs(Mstate);
}
}
C = (Efluct - Eavg*Eavg)/(Nsites*Nsites*kB*T*T);
fp << setw(20) << T << setw(20) << Eavg << setw(20) << C << setw(20) << Mavg << "\n" << endl;
T+=0.05;
}
fp.close();

//Output Final Configuration
fp.open("equiplotT2.0.txt");
if( fp.fail() ) {cout << "Cannot open file" << endl; exit(0);}
for(i=0;i<Nsites;i++) {
for(j=0;j<Nsites;j++) {
fp << Lattice(i,j) << setw(20);}
fp << endl;}
fp.close();


}

//-------------nieghbSum-----------------
int neighbSum(arrayt<int>& Lattice, int rx, int ry){
int Nsites = Lattice.n1();
double ns = (Lattice(periodic(rx+1,Nsites),ry) + Lattice(periodic(rx-1,Nsites),ry) +
Lattice(rx,periodic(ry+1,Nsites)) + Lattice(rx,periodic(ry-1,Nsites)));
return ns;
}




//------- Code For Determining Equillibriation ---------------
//        fp.open("equivalT2.0.txt");
//        if( fp.fail() ) {cout << "Cannot open file" << endl; exit(0);}
```

```
//
//        if (k % (Nsites*Nsites) == 0){
//            Estate = 0.0;
//            for (i=0; i<Nsites-1; i++) {
//                for (j=0; j<Nsites-1; j++) {
//                    Estate += (-jExch * neighbSum(Lattice, i, j) * Lattice(i,j));
//                }}
//            fp << setw(20) << k << setw(20) << Estate << "\n" << endl;}
//        fp.close();


//------- Code For Displaying Final Configuration ---------------
// Output Final Configuration
//fp.open("equiplotT2.0.txt");
//if( fp.fail() ) {cout << "Cannot open file" << endl; exit(0);}
//    for(i=0;i<Nsites;i++) {
//        for(j=0;j<Nsites;j++) {
//            fp << Lattice(i,j) << setw(20);}
//        fp << endl;}
//    fp.close();


//------- Code For Displaying Initial Configuration ---------------
//fp.open("gridinit.txt");
//if( fp.fail() ) {cout << "Cannot open file" << endl; exit(0);}
//    for(i=0;i<Nsites;i++) {
//        for(j=0;j<Nsites;j++) {
//            fp << Lattice(i,j) << setw(20);}
//        fp << endl;}
//    fp.close();
```