Computational Physics Project - The Ising Model of a Ferromagnet

April 23, 2016

Abstract

Produced a computational simulation of a 2D ferromagnetic spin system using the Ising model. In particular, employed the Metropolis algorithm with Monte-Carlo techniques to a spin lattice with periodic boundary conditions. Calculated the average magnetisation, energy and heat capacity of the system after reaching thermal equilibrium for a range of initial conditions. A second order phase transition was observed, where the system exhibits spontaneous symmetry breaking. Found the critical temperature when this occurs at $T_c = 2.33 \pm 0.03$ J/k_B, which is consistent with the literature value from Onsager to within two σ.

Contents

1 Introduction

The problem I chose to investigate is the Ising Model of a Ferromagnet. The model, named after Physicist Ernst Ising, simplifies spin-spin interactions in a ferromagnetic material by forming a lattice of spins which can point up or down. Only interactions between nearest neighbour pairs of spins are considered, so for a square lattice with N^2 sites the energy takes the form

$$E = -J \sum_{<ij>} S_i S_j - \mu H \sum_{i=1}^{N^2} S_i \tag{1}$$

where J is the exchange energy, μ is the magnetic moment and H is the external magnetic field. When J $>$ 0 it is energetically favorable for the spins to align. The system is in contact with a heat bath of temperature T, so individual spins can flip back and forth by exchanging energy with the bath. [2]

The aim of the project is to simulate this system by creating a program in C++ and using it to calculate macroscopic physical quantities, such as the magnetisation, energy and heat capacity. It is then of interest to investigate how these quantities vary with different initial conditions. In the next section I discuss the computational details of the problem, leading onto a summary of the implementation and a brief look at performance. The results are presented and discussed before reaching conclusions on the topic, with the full program listing contained in the appendix.

2 Analysis

There are a vast number of microstates that the system can explore (2^{N*N}), so there is no hope of dealing with this many states for anything larger than a trivial N. The Metropolis-Hastings algorithm was used to simulate the 2D spin system efficiently. On thermodynamical grounds, the probability P of finding the system in a state with energy E is given by

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

The Metropolis algorithm says 'instead of choosing configurations randomly, then weighting them with exp(-E/k$_B$T), we choose configurations with probability exp(-E/k$_B$T) and weight them evenly.' [3]

Consider a spin system in a known microstate α with energy E$_\alpha$ and a nearby microstate β with energy E$_\beta$ (nearby in this instance means flipping a single spin). If ΔE = E$_\beta-$ E$_\alpha$ $<$ 0, transition to the new microstate (flip the spin), otherwise transition with probability p = exp(ΔE/k$_B$T). This gives a single Monte-Carlo iteration. Applying this to each spin in the system gives a single time step.

The system will start in an initial state which is unlikely to be thermodynamically favorable, therefore a large number of time steps have to be completed to reach thermodynamic equilibrium. Once this is reached, relevant physical quantities can be sampled from the system and averaged over additional iterations.

The algorithm makes physical sense because it applies the principle of detailed balance. This says that at thermal equilibrium, each elementary process (in this case the flipping of spins) is equilibrated by its reverse process. There is no net change in the system else it would not be in thermal equilibrium.

I decided that I would start with the initial configuration of spins all pointing up. It would also be possible to start with some random initial configuration, and on the grounds of intuition it seems that this configuration would be faster to converge to thermal equilibrium. However, starting with all spins aligned makes it much easier to test the program is performing as expected, and it could easily be changed retrospectively.

Similarly, I chose to iterate sequentially over the whole lattice in a single time step. The alternative is to randomly sample a location in the lattice, and repeat this until an appreciable fraction of the whole lattice has been visited. I chose the former on the grounds of ease of testing.

Since the lattice is of finite size, periodic boundary conditions are used to avoid edge effects. Scaling the problem up to higher dimensions or adding more interactions between spins is straightforward with the Ising model, as this simply involves summing over more terms in equation (1).

3 Implementation

My overall approach to the problem was to create a representation of the spins, and iterate over the whole lattice with Monte-Carlo methods a large number of times until thermal equilibrium is reached. We proceed to iterate over the lattice again, but now at the same time sampling the magnetisation, energy and heat capacity after each time step. Finally the average value per lattice site of these quantities is calculated and output.

The 2D lattice of spins was represented by a 1D array of type signed char. I could then represent spin up as +1 and spin down as -1. A 1D array can be used to represent a 2D square lattice thanks to the modulo operator, and by taking care with array indexing. It is much easier than implementing a 2D array, which requires multiple loops to visit all the sites.

The algorithm *nearest_neighbour* took care of the indexing problems at the edges of the lattice by finding the locations of the 4 nearest neighbours using periodic boundary conditions. I only used expressions for the general case of a lattice of size N, where N was a global constant. This meant it was trivial to change the value of N before program compilation.

Perhaps the most important algorithm in the program, as it can be called more than 10^6 times, is the Monte-Carlo algorithm *metropolis*. This performs the Metropolis algorithm as explained above, but of note is the source of the random numbers used to decide whether to flip the spins when $\Delta E > 0$. The program uses the GNU scientific library (GSL) algorithm *gsl_rng*. The algorithm is seeded from the processor clock so that it generates a different sequence each time the program is run. The particular algorithm used 'has a Mersenne prime period of 2^{19937} - 1 and is equi-distributed in 623 dimensions.' [4]

4 Performance

From a theoretical point of view, using big O notation, the program scales as O(n), where n is the number of Monte-Carlo iterations performed. This is because for each iteration the complexity is O(1), and we repeat this n times. In terms of the lattice size N and the number of time steps j, the program scales as O(N^2j). This means there is a somewhat heavy cost in increasing N.

I have tried to minimise memory usage as much as possible. A signed char is one of the smallest possible data types at one byte. I used this data type to represent the spins as it is easily converted to a double or int. Flipping the spins is simply done by multiplying by -1. Only the current state of the system is stored so the memory usage remains constant throughout runtime.

As an indication of actual CPU runtime, it took 552 seconds to perform $\sim 4 \times 10^{-9}$ iterations in one calculation.

5 Results and Discussion

After creating the base code, some tests were performed to ensure that the program was working correctly. Initialising the spins to all point up, a single time step was performed and the state of the system was output for a variety of starting temperatures with no external magnetic field. The system behaved as expected; almost all the spins remained aligned for low temperatures ($\sim$1 J/k_B), as they did not have enough energy to explore other microstates. Almost all of the spins flipped for high temperatures ($\sim$10 J/k_B), because there is an excess of energy in the system. Between these two extremes an intermediate behaviour was seen, with only a fraction of the spins flipping.

Now I had ascertained that the program behaved correctly for a single time step, I added in a loop to perform many time steps (typically $\sim 1 \times 10^3$) with the aim of reaching thermal equilibrium. The program would then continue to iterate whilst also sampling the energy, magnetisation and heat capacity before outputting the average values after many iterations.

The values of these quantities also behaved as expected thermodynamically, as seen in figures 1, 2 and 3. The energy increased in some non linear way with temperature. At low temperatures there was a large magnetisation, as spins do not have enough energy to explore other microstates. High temperatures cause the magnetisation to fall to zero, as each microstate is equally likely and the energy reduction from aligning spins is negligible.

Between these two extremes we observe a second order phase transition in the system. There is strong ferromagnetism at low temperatures, as shown by the large value of the magnetisation per site. However, this sharply transitions to a regime with no net magnetisation. The temperature at which this occurs

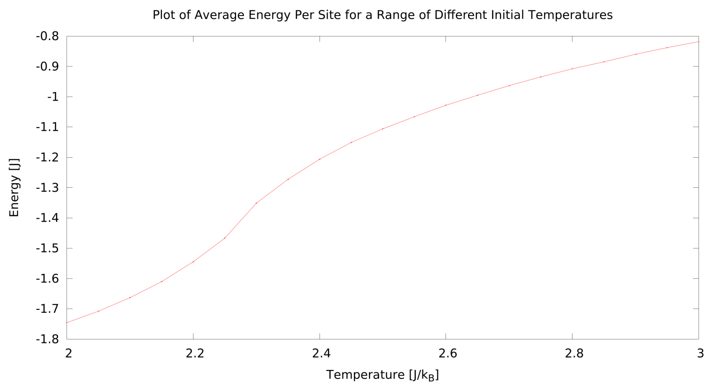


Figure 1: Average energy per site in the lattice for temperatures between 2 J/k$_B$ and 3 J/k$_B$, increasing in intervals of 0.05 J/k$_B$. Calculated with 2.5×10^3 iterations, and lattice size N = 100. There is a general trend for the energy to increase with temperature, but it is clearly non linear. This is thanks to the second order phase transition.

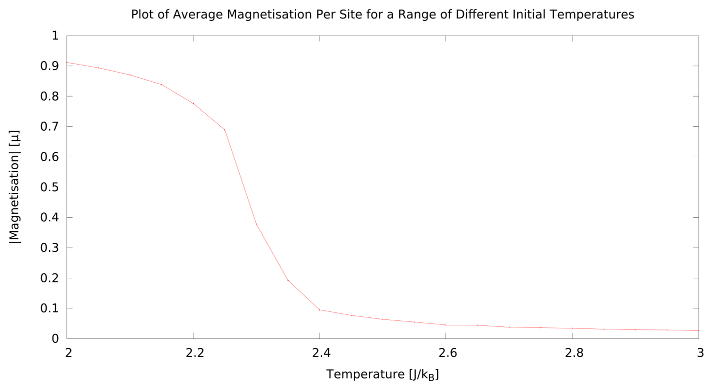


Figure 2: Average magnetisation per lattice site for temperatures between 2 J/k$_B$ and 3 J/k$_B$, increasing in intervals of 0.05 J/k$_B$. Calculated with 2.5×10^3 iterations, N = 100. The magnetisation is $\sim 1\ \mu$ for low temperatures until it gets close to T$_c$, then sharply falls to 0.

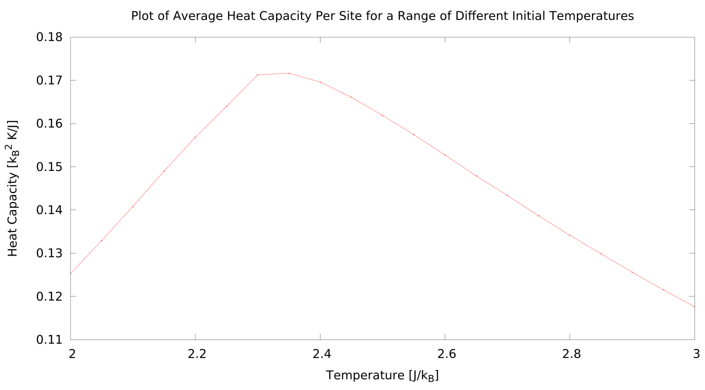


Figure 3: Average heat capacity per lattice site for temperatures between 2 J/k$_B$ and 3 J/k$_B$, increasing in intervals of 0.05 J/k$_B$. Calculated with 2.5×10^3 iterations, N = 100. The heat capacity increase as the temperature approaches the critical temperature, reaching a maximum when T = T$_c$.

is known as the critical temperature T$_c$. This transition also has an effect on the energy, but it is more easy to see in the heat capacity where the peak value corresponds to the critical temperature. The fluctuation-dissipation theorem allows us to calculate the heat capacity for a system in thermal equilibrium. It says that

$$C = \frac{\sigma_E^2}{k_B T^2} \tag{3}$$

where σ_E is the standard deviation of fluctuations in the energy of the system.

The phase transition is second order because there is no latent heat released; the energy is continuous across the transition. If we start in a disordered high temperature state and proceed to decrease the temperature through T$_c$ we acquire a net magnetisation. Crucially, the magnetisation has an orientation, but there was initially no favorable direction for the system to choose. The symmetry of the system is spontaneously broken at T$_c$, as it is forced to acquire an orientation.

Lars Onsager found an analytic result for the critical temperature in the specific case of a two dimensional square lattice.

$$T_c = \frac{2}{\ln(1+\sqrt{2})} \approx 2.269\ J/k_B \tag{4}$$

Compare this to the value obtained in figure 3, which gave T$_c$ = 2.35 ± 0.05 J/k$_B$.

Since it is now possible to identify the critical temperature, the next logical step is to look at how T$_c$ is affected by varying the size of the lattice, N. The theory of finite size scaling says that the critical temperature for a lattice of size N^2 varies as

$$T_c(N^2) = T_c(\infty) + aN^{-2/\nu} \tag{5}$$

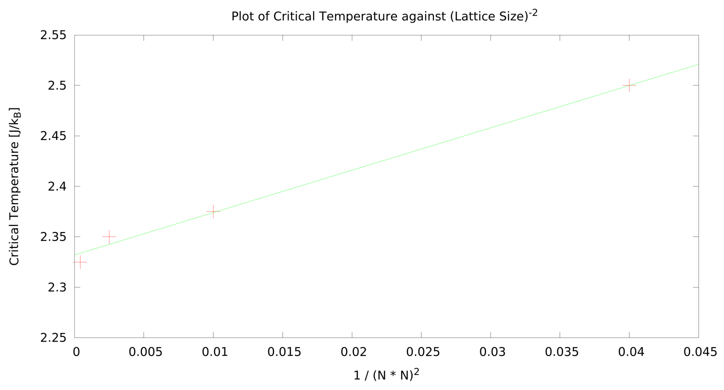


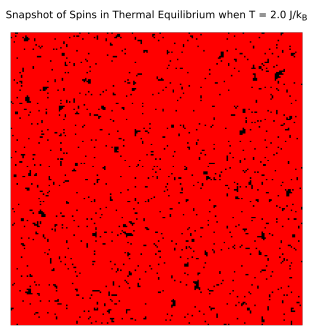
Figure 4: Used finite size scaling with the literature value $\nu = 1$ to find T$_c$ more accurately. The heat capacity is only maximised at the critical temperature in the limit as N $\rightarrow \infty$. Plot has a gradient of a = 4.2 $\pm$ 0.3, and gives a value of T$_c$ = 2.33 $\pm$ 0.03 J/k$_B$. This is higher than the theoretical value, but can be explained by the limited CPU power available. Number of iterations, j = 5×10^4.

ν has a predicted value of 1, so plotting T$_c$ against N^{-2} should produce T$_c$ as the y intercept. This value can then be compared with Onsager's analytical calculation, (4). The results are shown in figure 4. Computational time scales as N^2 so it is very time consuming to use large values of N, which limited the accuracy of the results greatly. It is possible to reduce the number of iterations before calculating quantities, but if this is reduced too much the system is no longer in thermal equilibrium and the accuracy suffers again.

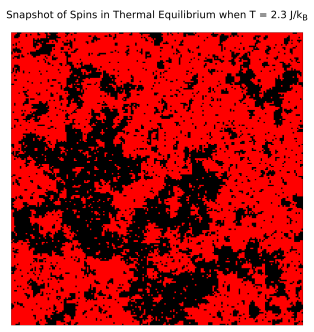
The temperature at which the maximum value of the heat capacity occurs was taken as T$_c$, which meant there was an additional error in identifying this from the data. Due to CPU runtime constraints, I only had time to increment the temperature over the same value of N every ΔT = 0.025 J/k$_B$. This is of the same order of magnitude as the changes in T$_c$, which explains the slightly poor fit. It also explains why we calculate a value of T$_c$ = 2.33 $\pm$ 0.03 J/k$_B$, larger than the theoretical value, but still in agreement.

After looking at how the average values of the macroscopic physical quantities vary with initial conditions, it is insightful to look at the microscopic behaviour of the spins between time steps. Snapshots in time of the spin states after reaching thermal equilibrium for a range of temperatures are illustrated in figure 5. I have also produced some gifs to show how the microstructure evolves in time, and these are included in the material submitted online.

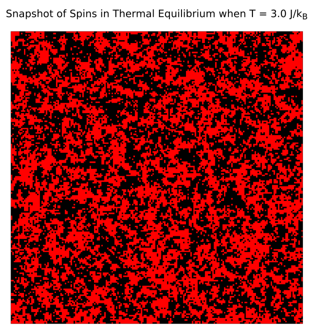
For temperatures below T$_c$, almost all of the spins are aligned in the same direction with only a few very small clusters of misaligned spins. Looking at how the spins vary with time, most of the spins remain in the spin up state. There are only a few spins flipping down at each iteration, but these proceed to



(a) At temperatures less than T$_c$ almost all of the spins are aligned



(b) Large domains form close to T$_c$, as the symmetry of the system is broken.



(c) The system is highly disorder for temperatures above T$_c$, with only small domains forming.

Figure 5: A set of microstates explored by the spin system after thermal equilibrium has been reached. Red is a spin up state, black is spin down. Produced after 1×10^3 iterations, with N = 200.

flip back up in the next time step.

Close to T$_c$, large magnetic domains form. These domains are largest at the critical temperature, and they do not remain constant in size or position as the system is evolved forwards in time. The system has sufficient thermal energy to explore new microstates, leading to large clusters of spins in the same orientations forming.

At temperatures significantly above T$_c$, the system is very disordered with lots of very small domains. There is no net magnetisation and after each time step almost all of the spins flip. The high thermal energy of the system means that the energy cost of misaligning neighbouring spins is negligible.

If we set the interaction energy in (1) so that $J < 0$, we observe antiferromagnetism. This means it is favourable for spins to be misaligned with their neighbours. As can be seen in figure 6, small clusters of aligned spins form, but the majority of the system sits in a misaligned state.

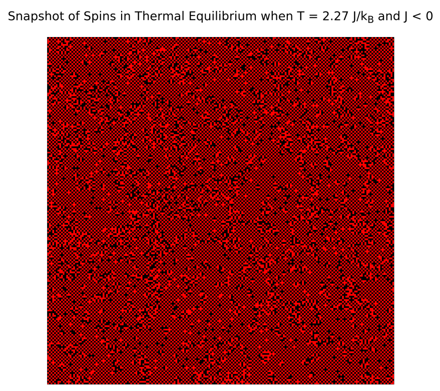
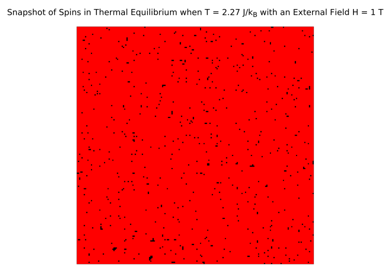
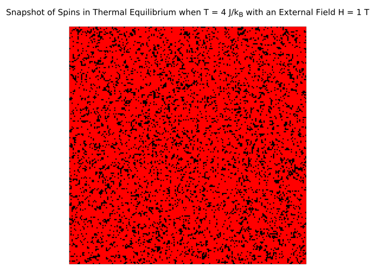


Figure 6: A specific microstate the system explored after reaching thermal equilibrium. This is an anti-ferromagnetic system as J $<$ 0. The majority of the system is in a misaligned state. Red is a spin up state, black is spin down. Produced after 1×10^3 iterations, with N = 200.



(a) At T$_c$ there are not any large domains, thanks to the external field.



(b) Even above T$_c$ only small domains form, and there is still a net magnetisation.

Figure 7: A set of microstates the system explored after reaching thermal equilibrium with an external magnetic field H = 1 T. Red is a spin up state, black is spin down. Produced after 1×10^3 iterations, with N = 200.

Finally, it is possible to look at the effect on applying an external magnetic field to the system. With no external field there is no preferred direction (the initial configuration does not matter when the system is in thermal equilibrium after a large number of iterations). When an external field is switched on, it is energetically favorable for the spins to align with the field. Due to this extra term in the energy from (1), large domains are no longer formed at T$_c$. We can also see from figure 7 that at temperatures above T$_c$ the system is much more resistant to forming domains than when H = 0.

6 Conclusions

I have been successful in simulating a 2D ferromagnetic spin system through the use of the Ising model and its implementation via the Metropolis algorithm. The time evolution of the microstates of the system was observed, and macroscopic physical quantities calculated after reaching thermal equilibrium. These values have the same dependence on temperature that we would expect from thermodynamic arguments.

Of particular interest in the 2D spin system with no external field is the second order phase transition, where we observe spontaneous symmetry breaking. The temperature at which this occurs, the critical temperature, was calculated to be $T_c = 2.33 \pm 0.03$ J/k$_B$. This value was obtained using finite size scaling (5), and by taking the critical temperature for each system to be when the heat capacity is at a maximum. This is in agreement with the theoretical value produced by Onsager (4), $T_c \approx 2.269$ J/k$_B$. The accuracy of the computational value could be improved greatly by spending more CPU time to reduce the temperature increments Δ T.

References

[1] David Buscher *Part II Computational Physics* Course Handouts: Department of Physics, University of Cambridge

[2] *https://en.wikipedia.org/wiki/Ising_model*; April 2016

[3] Metropolis, Nicholas, et al. *Equation of state calculations by fast computing machines* J. Chem. Phys. 21.6 (1953): 1087-1092.

[4] *https://www.gnu.org/software/gsl/manual/html_node/Random-number-generator-algorithms.html*; April 2016

A Full Program Listing

```
//Use Metropolis algorithm to investigate ferromagnetism using the Ising
//model. Periodic boundary conditions used so there are no edge effects

#include <iostream>
#include <cmath>
#include <gsl/gsl_rng.h>
using namespace std;

//set up constants
int const N = 200; //Size of the lattice (N x N)
double const mu = 1; //Magnetic moment
double const H = 0; //External magnetic field
double const J = 1; //Exchange energy, set to 1 for simplicity

signed char metropolis (signed char n1, signed char n2, signed char n3,
        signed char n4, signed char s, gsl_rng *rng, double T) {

        //signed chars are 4 nearest neighbours + current value of spin
        double dE; //change in energy if spin is reversed

        //Find change in energy from Ising model,
        //E = - J sum <ij> {Si Sj} - mu * H sum i {Si}
        dE = J * (n1 + n2 + n3 + n4) * (2 * s) + mu * H * (2 * s);

        //flip spin if dE < 0 or exp( - dE / (k_B * T)) > p,
        //for p randomly sampled in [0,1]
        if (dE < 0)
                return (-1 * s);

        else {
                double p = exp(-dE / T);

                if (p > gsl_rng_uniform(rng))
                        return (-1 * s);

                else
                        return s;
        }
}

void nearest_neighbour (int n[4], int i) {
        //find position of the nearest neighbours for a given index

        if (i == 0)  {
        //if on the top left corner of the lattice
                n[0] = i + 1; n[1] =  N - 1; n[2] = N * (N - 1);
                n[3] =  N;
        }

        else if (i == (N * (N - 1))) {
                //if on bottom left corner
                n[0] = i + 1; n[1] = N * (N - 1); n[2] = i - N;
                n[3] = 0;
        }

        else if ((i % N) == 0) {
                //if on the left edge & not in a corner
                n[0] = i + 1; n[1] = i + (N - 1); n[2] = i - N;
                n[3] = i + N;
        }

        else if (i == (N - 1)) {
                //if on top right corner
                n[0] = N; n[1] = i - 1; n[2] = N * N - 1;
                n[3] = i + N;
        }

        else if (i == (N * N - 1)) {
                //if on bottom right corner
                n[0] = N * (N - 1); n[1] = i - 1; n[2] = i - N;
                n[3] = N - 1;
        }

        else if ((i % N) == (N - 1)) {
                //if on right edge & not in corner
                n[0] = i - (N - 1); n[1] = i - 1; n[2] = i - N;
                n[3] = i + N;
        }

        else if (i < N) {
                //if on top edge & not in a corner
                n[0] = i + 1; n[1] = i - 1; n[2] = i + N * (N - 1);
                n[3] = i + N;
        }

        else if (i > N * (N - 1)) {
                //if on bottom edge & not in a corner
                n[0] = i + 1; n[1] = i - 1; n[2] = i - N;
                n[3] = i % N;
        }

        else {
                n[0] = i + 1; n[1] = i - 1; n[2] = i - N;
                n[3] = i + N;
        }
}

double energy (signed char spins[N * N], double &var) {
        //calculate sum E and sum E^2

        double E = 0, E2 = 0, tmp;
        int n[4] = {}; //array to pass values of nearest neighbours

        for (int i = 0; i < N * N; i++) {

                nearest_neighbour(n, i);
                tmp = - J * spins[i] * (spins[n[0]] + spins[n[1]]
                + spins[n[2]] + spins[n[3]]) - mu * H * spins[i];
                E += tmp;
                E2 += tmp * tmp;
        }

        tmp = E / (N * N);
        var = (E2 / (N * N) - tmp * tmp) / (N * N - 1);

        return tmp;
}

double magnetisation (signed char spins[N * N]) {
        //Magnetisation is given by the average spin value;
        //M = 1/N sum i {Si}

        double M = 0;

        for (int i = 0; i < N * N; i++) {

                M += (double) spins[i];
        }

        return (M / (N * N));
}

void iterate (signed char spins[N * N], gsl_rng *rng, double T) {
        int n[4] = {}; //array to pass values of nearest neighbours

        //Iterate over the whole lattice, applying Metropolis algorithm
        //to determine whether to flip spin or not
        for (int i = 0; i < N * N; i++) {

                nearest_neighbour(n, i);
                spins[i] = metropolis(spins[n[0]], spins[n[1]],
                spins[n[2]], spins[n[3]], spins[i], rng, T);
        }
}

void output(signed char spins[N * N]) {
        //outputs the current state of the lattice in a format to
        //produce a 2d heat map

        for (int i = 0; i < N * N; i++) {

                cout << (int) (i / N) << "␣" << i % N << "␣"
                << (int)spins[i] << endl;

                if (i % N == (N - 1))
                        cout << endl;
        }
}

void equilibrium(signed char spins[N * N], gsl_rng *rng, double j,
        double T) {
        //iterate 2 * j times to reach equilibrium,
        //then a further j times to calculate properties of the system

        int n = 0; //counter for the number of iterations
        double m = 0, e = 0; //running total of magnetisation/energy
        //running total of the variance used to calculate heat capacity
        double var = 0;
        double tmp;

        while (n < (int)(2 * j)) {

                iterate(spins, rng, T);
                n++;
        }

        n = 0;

        while (n < (int)(j)) {

                m += magnetisation(spins);
                e += energy(spins, tmp);
                var += tmp;
                iterate(spins, rng, T);
                n++;
        }

        //output average magnetisation/energy/heat capacity per site
        //average heat capacity per site,
        //C = sigma^2 / (T^2) in units k_B ^ 2 * K / J
        cout << N << "␣" << T << "␣" << abs(m / j) << "␣" << (e / j)
        << "␣" << (var / (j * T * T)) << endl;


        //output(spins);
}

int main () {

        //Spins represented by signed char, 1 = up, -1 = down
        signed char spins[N * N];

        double Tmax = 4.1E0; // maximum temperature, units J / k_B
        double Tmin = 4E0; //minimum temperature to start loop from
        double dT = 2.0; //step size
        double T = Tmin; //Current temperature


        //initialise and seed rng
        gsl_rng *rng = gsl_rng_alloc(gsl_rng_default);
        gsl_rng_set(rng, time(NULL));

        double j = 1E3; //number of iterations

        //calculate properties of the system after equilibrium is
        //reached for a range of temperatures
        while (T < Tmax) {

                //Initialize lattice spins to all point up
                for (int i = 0; i < N * N; i++)
                        spins[i] = 1;

                equilibrium(spins, rng, j, T);
                T += dT;
        }

        gsl_rng_free(rng);

        //output running time
        //cout << (clock()/ CLOCKS_PER_SEC) << endl;

        return 0;
}
```