# Analysis of the Ising model

Exercise 2.3 and 2.4, Modelling and Simulation

Jim Carstens (5558816)

16th May 2019

Universiteit Utrecht

## Contents

## Introduction

In this report, the Ising model will be investigated using both the Metropolis algorithm and the Wolff algorithm. The Ising model is an important mathematical model which can be used to describe physical systems such as magnets and grids of correlated spins and there are even applications in neuroscience.

For a two-dimensional Ising model, the average magnetisation, energy, heat capacity, magnetic susceptibility and autocorrelation functions are investigated using a Monte Carlo model. Two different algorithms for implementing the Ising model are tested and compared: the Metropolis algorithm and the Wolff algorithm.

## 1 Theory

The Ising model describes a system of correlated spin states in a lattice. This correlation is given by the Hamiltonian of the system:

$$H = -J \sum_{\langle ij \rangle} s_i s_j - B \sum_i s_i. \tag{1}$$

In this equation, the first sum goes over $\langle ij \rangle$, which means the nearest neighbours $j$ of particle $i$, $J$ is the alignment energy and $B$ is the strength of an external magnetic field. Spins $s_i$ can take on a value of either $-1$ or $+1$. By choosing $J$ to be positive, energy is lowered by aligning spins, resulting in ferromagnetic behaviour. On the contrary, choosing $J$ to be negative causes the energy to be lowered by anti-aligning spins. This results in anti-ferromagnetic behaviour. In this report positive $J$ is considered. Moreover, a model with no external field is examined, so $B = 0$. For the phenomena investigated here an external field is not necessary. However, adding it to the model would not be a complicated undertaking.

One can look at many different dimensionalities for the Ising model. The one-dimensional model was solved exactly in 1925 by Ising [1]. In 1944, Onsager arrived at a mathematical solution for the 2D-model without an external magnetic field [2]. No solution has been found for the 3D-model so far, nor for higher dimensions. Numerically, as well as analytically with the right approximations however, many research as been done and much is known about the Ising model in these conditions. In this report, the two-dimensional Ising model is chosen to investigate.

An interesing property of this 2D-model is the presence of a phase transition, which the 1D-version does not have. The phase transition, which happens at the critical temperature $T_C$ divides the behaviour of the system in two regimes. Below $T_C$, the thermal fluctuations are not so large and spontaneous magnetisation occurs. Here the system is ferromagnetic. Above $T_C$, the thermal fluctuations are large enough to destroy the magnetisation of the system.

Several properties are interesting to measure. The mean magnetisation of the system is simply a sum of all the spins in the system:

$$\langle M \rangle = \left\langle \sum_i s_i \right\rangle. \tag{2}$$

The energy of the system can be calculated using the Hamiltonian of Equation 1. For both quantities it is also interesting to divide them by the amount of lattice points $N$. In the case of the magnetisation, this gives $\langle m \rangle = \langle M \rangle / N$. From these quantities, two more properties of the system can be calculated, the heat capacity

$$c_S = \frac{\beta^2}{N} \left( \langle E^2 \rangle - \langle E \rangle^2 \right), \tag{3}$$

and the magnetic susceptibility

$$\chi_m = \beta N \left( \langle M^2 \rangle - \langle M \rangle^2 \right), \tag{4}$$

where $\beta = \frac{1}{k_B T}$. In this report, $k_B$ is taken to be 1.

The autocorrelation of a certain quantity can be calculated as follows [3]:

$$C_M(t) = \frac{1}{t_m ax - t} \left( \sum_{t'=0}^{t_{max}-t} M(t') \times M(t'+t) - \sum_{t'=0}^{t_{max}-t} M(t') \times \sum_{t'=0}^{t_{max}-t} M(t'+t) \right). \tag{5}$$

In this formula specifically $C_M$, the magnetic autocorrelation function is given. By switching $M$ for another quantity, other autocorrelation functions can be computed.

## 2 Simulation

For the simulation, a program is made in C++ that generates a two-dimensional square lattice of width $L$ with total system size $N = L \times L$. This lattice is represented by an array containing `int` values of $\pm 1$, where the positive value arbitrarily corresponds to spin-up, and the negative value to spin-down. The system can be initialised either to a temperature $T = 0$, where all spins point in the same direction, or to $T = \infty$, where all spins are randomly chosen. For generating random numbers the generator `std::mt19937` is used with as seed the current time `std::chrono::steady_clock::now().time_since_epoch().count()`. This provides quick, highly random numbers.

Since the lattice is not of infinite size (this would require infinite computational power), periodic boundary conditions are used. This means that each site at the edge of the lattice has the other side of the lattice as its neighbours, as can be seen in Figure 1. Another approach that can be

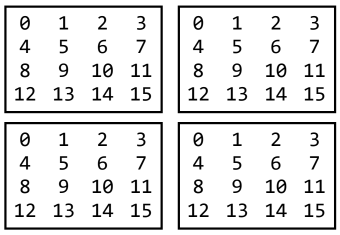


**Figure 1** – Periodic boundary conditions

used are helical boundary conditions, which are much faster to compute. There, the 2D-lattice is reduced to a 1D-lattice that is wrapped around itself. The neighbours can be calculated using arithmetic. A way to imagine this is by thinking about a film roll wrapped around a cylinder, where aligned frames are neighbours.

In this report, periodic boundary conditions are used because it allows for legible code. Moreover, a variant on the normal Ising model is considered here. Normally the nearest neighbours $\langle ij \rangle$ of a spin that are considered are the four neighbours left, right, up and down. Here however, also the corner neighbours are considered. This brings the amount of neighbours to 8. For example, for site 10, the neighbours are $(5, 6, 7, 9, 11, 13, 14, 15)$.

When running the program, first, the system is started out at $T = 0$, or $T = \infty$ and allowed to equilibrate to the composition correspondign to the actual temperature that is set. Then, for many Monte Carlo steps, the spins are flipped according to an algorithm, the various observables are measure using equations 1 and 2. Then, the temperature can be changed and the previous is repeated. For each measurement of the energy and the magnetisation, 100 samples are taken at intervals of 100, after the system has been at a certain temperature for 1000 Monte Carlo steps. Later, this will prove to be sensible.

## 3 Metropolis algorithm

### 3.1 Method

The Metropolis algorithm works by flipping single flips and accepting a move according to the acceptance rule. This is also called single flip dynamics. This acceptance rule is as follows:

$$A(\mu \to \nu) = \begin{cases} e^{\beta \Delta E} & \text{if} \Delta E > 0 \\ 1 & \text{if} \Delta \leq 0 \end{cases} \tag{6}$$

For each Monte Carlo move, $N$ single flips are attempted. Thus, on average, each site will be visited during each move. Because a single flip can be undone with an equal chance, this algorithm preserves ergodicity.

The measurements are performed by at the beginning of the simulation computing the energy and magnetisation for the whole system, and from then on adding the difference, to save computation time. Here

$$\Delta E = 2Js_k^\mu \sum_{\langle ij \rangle} s_i^\mu, \tag{7}$$

and

$$\Delta M = 2s_k^v \tag{8}$$

are used. The other quantities can be calculated from the former. The magnetization autocorrelation function is calculated after the simulation, by looking at 1000 consecutive measurements for the magnetisation right after setting the temperature. Then, in Wolfram Mathematica, the following formula is used:

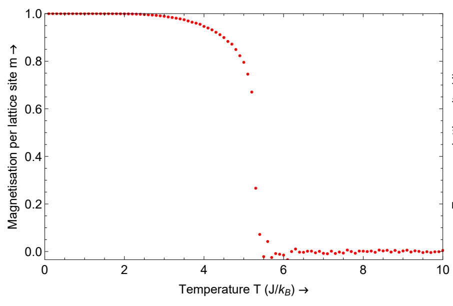
```
mac = Table[1/(tmax - t)*Sum[mdat[[t2]]*mdat[[t2 + t]], {t2, 1, tmax - t}] -
Sum[mdat[[t2]], {t2, 1, tmax - t}] *
Sum[mdat[[t2 + t]], {t2, 1, tmax - t}], {t, 1, tmax - 1}];
```

The acceptance values are pre-calculated and stored in an acceptance array after setting the temperature, to avoid needing to calculate these exponents every spin swap step.

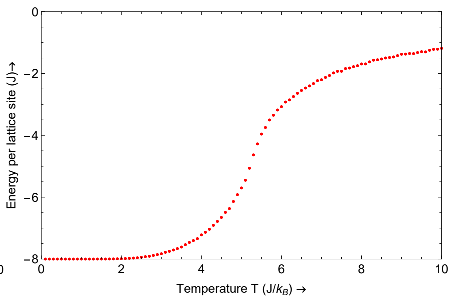
The code that was used can be seen in Appendix A.

## 3.2 Results

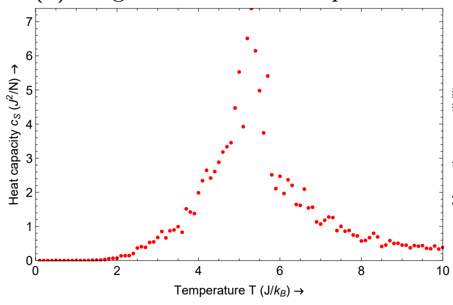
In Figure 4, the results of calculating various quantities along a range of temperatures are displayed, for the energy and the magnetisation with errors. However, these errors are so small that they are not visible. It can be seen that at low temperatures, the system is fully magnetised, corresponding to the lowest possible energy $E = 8J/\text{spin}$, which is the amount of neighbours a spin has. Upon increasing temperature, not much happens, until the critical temperature is reached. There, the magnetisation drops to zero, and the enery also goes to zero. The thermal fluctuations overpowerd the paramagnetic tendencies of the material. In the program, this is a result from the fact that at high temperature, many moves that rise the energy are also accepted.



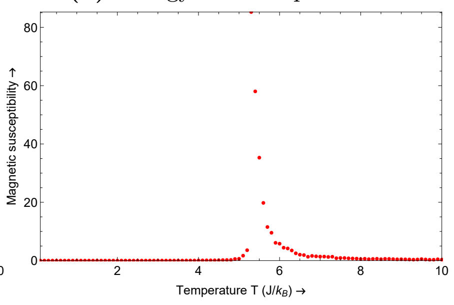
**(a)** Magnetisation over temperature



**(b)** Energy over temperature



**(c)** Heat capacity over temperature



**(d)** Magnetic susceptibility over $T$

**Figure 2** – Plots of the magnetisation(including errors), energy (including errors), heat capacity and magnetic susceptibility against temperature. All values are per lattice site for clarity.

The heat capacity and the magnetic susceptibility obtain a very high value around the critical temperature. The critical temperature is estimated for this system to be $T_C = 5.3J$. No theoretical background on the eight-neighbour Ising model was found to compare this value to.

In Figure 3, it can be seen that the magnetic autocorrelation is very stable before $T_C$ and much less after $T_C$.

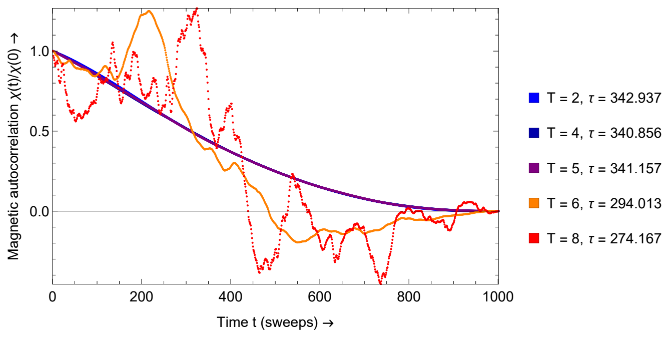


**Figure 3** – The magnetic autocorrelation function $C_M$ calculated for various temperature. The correlation time is displayed in the legend.

## 4 Wolff algorithm

### 4.1 Method

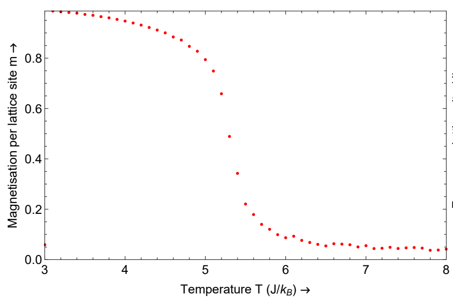
Wolff's algorithm is used to overcome the long correlation times of the Metropolis algorithm around the critical point. It works by finding clusters of same spins and flipping them all according to a certain probability. Spins are added to the cluster according to the following probability

$$P_{add} = 1 - e^{-2\beta J}. \tag{9}$$

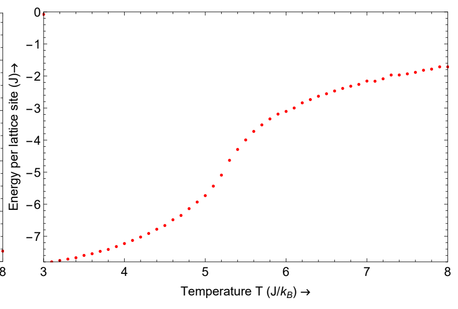
The code that was used can be seen in Appendix B.

### 4.2 Results

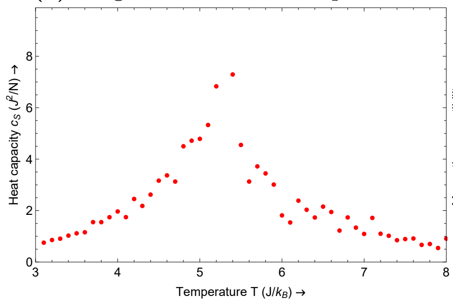
In Figure 4 the results for using the Wolff algorithm can be seen. In general, all results are rather similar to the results of the Metropolis algorithm. The difference is that the focus is more made on the region of the phase transition. Also, for the magnetic susceptibility, the divergence of the peak at $T_C$ is less extreme with the Wolff algorithm than with the Metropolis algorithm. The value for the critical temperature still seems to be $T_C = 5.3J$.



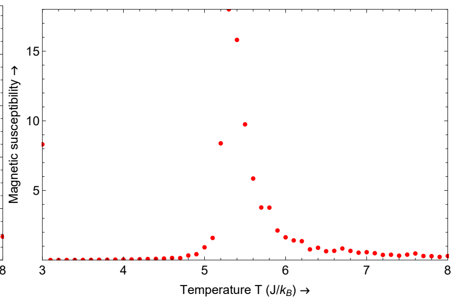
**(a)** Magnetisation over temperature



**(b)** Energy over temperature



**(c)** Heat capacity over temperature



**(d)** Magnetic susceptibility over $T$

**Figure 4** – Plots of the magnetisation(including errors), energy (including errors), heat capacity and magnetic susceptibility against temperature. All values are per lattice site for clarity.

There is however a clear difference at the autocorrelation function. As can be seen in Figure 5, the autocorrelation function remains neat across all temperatures, even very close to the critical temperature. A qualitative observation is that after the critical temperature, the algorithm runs very fast. This is probably because there is so much thermal nosie that no big clusters can be formed.

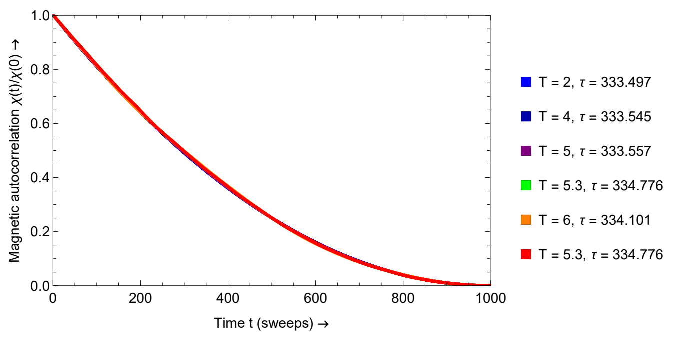


**Figure 5** – The energy autocorrelation function $C_E$ calculated for various temperature. The correlation time is displayed in the legend.

## 5 Further information

Plot showing the equilibation of the various measured quantities can also be made using the C++ code of the simulation. Moreover, using a Python file and the `write()` function of the simulation, videos of the Ising model evolving can be made. Upon request these can be provided by the author.

## References

[1] Ernst Ising. 'Contribution to the Theory of Ferromagnetism'. In: *Z. Phys.* 31 (1925), pp. 253–258. DOI: 10.1007/BF02980577.

[2] L. Onsager. 'Crystal Statistics. I. A Two-Dimensional Model with an Order-Disorder Transition'. In: *Physical Review* 65 (Feb. 1944), pp. 117–149. DOI: 10.1103/PhysRev.65.117.

[3] M Newman and G Barkema. *Monte carlo methods in statistical physics chapter 1-4*. Oxford University Press: New York, USA, 1999.

## A Metropolis algorithm

The following implementation of the Metropolis algorithm that runs every Monte Carlo step, was used.

```cpp
void sweep() {
  for (int i = 0; i < lattice_size; i++) {
    int random_site = std::uniform_int_distribution<int>(0, lattice_size - 1)(rng);
    int random_x = random_site % N;
    int random_y = random_site / N;
    int sum = 0;
    //for (int n : {i - N - 1, i - N, i - N + 1, i - 1, i + 1, i + N - 1, i + N, i +
    //above could be faster, but checking periodic boundary cond's becomes messy.
    for (int dx = -1; dx < 2; dx++) {
      for (int dy = -1; dy < 2; dy++) {
        if (dx == 0 && dy == 0) continue;
        int neighbour_x = random_x + dx;
        int neighbour_y = random_y + dy;
        if (neighbour_x == N) {
          neighbour_x = 0;
        }
        if (neighbour_x == -1) {
          neighbour_x = N - 1;
        }
        if (neighbour_y == N) {
          neighbour_y = 0;
        }
        if (neighbour_y == -1) {
          neighbour_y = N - 1;
        }
        sum += lat(neighbour_x, neighbour_y);
      }
    }
    int delta_E = 2 * J * lat(random_x, random_y) * sum;
    if (delta_E <= 0) {
      flip(random_x, random_y);
      E_total += 2 * delta_E;
      M_total += 2 * lat(random_x, random_y);
    }
    else if (random_zero_one(rng) < prob[delta_E]) {
      flip(random_x, random_y);
      E_total += 2 * delta_E;
      M_total += 2 * lat(random_x, random_y);
    }
  }
}
```

## B Wolff algorithm

Below can be seen the implementation of the Wolff algorithm, that manipulates the spins at each Monte Carlo simulation step.

```
void wolff_step() {
  int stack[lattice_size];
  int seed = std::uniform_int_distribution<int>(0, lattice_size - 1)(rng);
  int seed_x = seed % N;
  int seed_y = seed / N;
  stack[0] = seed;
  int sp = 1; // sp is the size of the LIFO buffer
  int oldspin = lat.at(seed);
  int newspin = -lat.at(seed);
  flip(seed_x, seed_y);
  while (sp) {
    int current = stack[--sp];
    int current_x = current % N;
    int current_y = current / N;
    for (int dx = -1; dx < 2; dx++) {
      for (int dy = -1; dy < 2; dy++) {
        if (dx == 0 && dy == 0) continue;
        int neighbour_x = current_x + dx;
        int neighbour_y = current_y + dy;
        //periodic BC's
        if (neighbour_x == N) neighbour_x = 0;
        if (neighbour_x == -1) neighbour_x = N - 1;
        if (neighbour_y == N) neighbour_y = 0;
        if (neighbour_y == -1) neighbour_y = N - 1;
        if (lat(neighbour_x, neighbour_y) == oldspin) {
          //p_add = 1-exp(-2*beta*J)
          if (random_zero_one(rng) < p_add) {
            //add to cluster
            stack[sp++] = neighbour_x + N * neighbour_y;
            flip(neighbour_x, neighbour_y);
          }
        }
      }
    }
  }
}
```