The Ising model

Fast computation using the Metropolis Monte Carlo algorithm

Maarten L. Terpstra – s2028980
Computing Science – Computational Science & Visualization
m.l.terpstra@student.rug.nl
April 26, 2016

Abstract

In this report, the theoretical foundation and the practical results of the Ising model are discussed. The main issues of gathering statistical information about an Ising model are discussed and a practical estimation of these statistics is provided in the form of the Metropolis-Monte Carlo algorithm.

1 Introduction

A world as we live in today does not exist without magnets. Besides that Earth itself is one huge magnets guarding us from space radiation and debris, magnets are used in a plethora of applications, ranging from navigation, food preparation, storing data, producing sound, driving trains and detecting diseases. While we know how to interact and manipulate the macroscopic properties of magnets, the underlying mathematical theory can help uncover more interesting non-trivial properties, which may allow us to derive even more applications of magnets. An example of such non-trivial property is that of an phase transition. A phase transition is what happens when a system goes to one phase to another – e.g. water that freezes at a certain temperature. What can happen with magnets during a phase transition is that spontaneous magnetization occurs, or stops occurring. A strong mathematic model can help us analyze a problem and achieve a deeper understanding of macroscopic and microscopic properties. One such model, the Ising model, arose from statistical mechanics and gives a mathematical basis for (ferro)magnetism.

1.1 The Ising Model

The Ising model a long-studied, simple lattice model to describe the macroscopic magnetic properties of a thermodynamic system. Recently [1], it was found to be an universal spin model, that even allows mapping of arbitrary problems to Ising models. This could help us to gain more insight in very hard, or NP-Complete, problems. It consists of a number of atoms with spins that can have a value of $\sigma_i = \pm 1$. These spins can interact with each other, modifying each others spins and changing the energy of the system. The energy is calculated according to the Hamiltonian

$$\mathcal{H} = -J\sum_{i,j}\sigma_i\sigma_j - H\sum_i \sigma_i$$

which gives us the total energy of the system. Here, J is the interaction constant, H is the magnetic moment – i.e. how a spin interacts with the external field – and i and j are neighbors. E refers to the energy of a specific configuration. The energy of a particular one-dimensional configuration S is given by

$$E = -\sum_{n=1}^{N-1} S_n S_{n+1}.$$

Note that this implement so called "free-edges" boundary conditions: elements at the boundary have no neighbor beyond the boundary. When $J > 0$, this means that the material is ferromagnetic. This means that the system "favors" configurations where all the spins point in the same direction – i.e. all configurations where i and j are neighbors, $\sigma_i = \sigma_j$ have a higher probability of occurring. When $J < 0$, the material

is anti-ferromagnetic – i.e. it favors configurations where neighboring spins are different than the spin of a particle.

For the remainder we use that $J = 1$ and $H = 0$ – i.e. a ferromagnetic material without an external field. The Ising model can be used to study how the system behaves as a function of the (thermodynamic) temperature T. Statistical mechanics tells us that the probability that a configuration c is realized with energy E follows from the probability distribution

$$P_c = \frac{e^{-\beta E_c}}{\sum\limits_{\{S_0,S_1,\ldots,S_n\}} e^{-\beta E}}$$

where $\beta = \frac{1}{T}$. The denominator – which is also called the partition function, often denoted by Z – sums over all possible configurations, i.e. $Z = e^{-\beta E_1} + e^{-\beta E_2} + \ldots$. The free energy of a system is thus the weighted average of all possible configurations $U = \sum_c E_c P_c = -\frac{\partial}{\partial \beta} \ln Z$. Since we know that $Z = (2\cosh\beta)^N$ we can express U in terms of β by

$$\begin{aligned}
U &= -\frac{\partial}{\partial \beta} \ln Z \\
&= -\frac{\partial}{\partial \beta} \ln \left(2^N \cosh^N \beta\right) \\
&= -\frac{\partial \ln u}{\partial u} \frac{\partial u}{\partial \beta} \quad \left(\text{with } u = 2^N \cosh^N \beta \text{ and } \frac{\partial \ln u}{\partial u} = \frac{1}{u}\right) \\
&= -\frac{\frac{\partial}{\partial \beta}\left(2^N \cosh^N \beta\right)}{2^N \cosh^N \beta} = -\frac{2^N \frac{\partial}{\partial \beta}\left(\cosh^N \beta\right)}{2^N \cosh^N \beta} \\
&= -\frac{\frac{\partial}{\partial \beta}\left(\cosh^N \beta\right)}{\cosh^N \beta} \\
&= -\frac{N \cosh^{N-1}(\beta) \frac{\partial}{\partial \beta} \cosh \beta}{\cosh^N \beta} \\
&= -\frac{N \frac{\partial}{\partial \beta} \cosh \beta}{\cosh \beta} \\
&= -N \frac{\sinh \beta}{\cosh \beta} = -N \tanh \beta.
\end{aligned}$$

Therefore, the average energy per spin $\frac{U}{N} = -\tanh\beta$. In the same way, we can derive an exact solution for the specific heat. In short, the specific heat C gives the fluctuation of the energy as a function of the temperature. By this description, we can determine that

$$C = \frac{\partial U}{\partial T} = -\frac{\beta}{T}\frac{\partial U}{\partial \beta} = -\beta^2 \frac{\partial U}{\partial \beta}.$$

Therefore,

$$C/N = -\frac{\beta^2}{N}\frac{\partial U}{\partial \beta} = \left(\frac{\beta}{\cosh \beta}\right)^2.$$

Now that it is known which macroscopic results are possible at a given temperature, it is not known *which* configurations achieve these results. One possibility is to try every possible configuration and evaluate P_c. Since every atom out of N atoms on the lattice can obtain two different spins, there are 2^N possible configurations. Computing this for a sufficiently large square lattice this would require an infeasible amount of configurations – i.e. a 20×20 lattice would evaluating 2^{400} configurations. Assuming one can compute 10^{12} sums per second – which is optimistic – this would still take several thousands of years. One way to circumvent this, is by using the Metropolis-Monte Carlo Algorithm.

1.2 Metropolis-Monte Carlo

Metropolis-Monte Carlo (hereafter, MMC) is an algorithm to generate configurations drawn from an unknown distribution. It obtains a sequence of random samples from a random distribution. With regard to the Ising model, it is a very simple algorithm:

procedure MMC(n)
  $S \leftarrow$ Random configuration
  $E \leftarrow E(S)$ ▷ i.e. E is the energy of the configuration
  while $i < n$ **do**
    $j \leftarrow random() \cdot n + 1$ ▷ $random() \in [0, 1)$
    $\delta_E \leftarrow$ change in energy if $S(j) = -S(j)$
    if $e^{-\beta\delta_E} > random()$ **then** ▷ Accept the spin flip
      $S(j) \leftarrow -S(j)$
      $E \leftarrow E + \delta_E$
    end if
    $i \leftarrow i + 1$
  end while
  return E
end procedure

A new state is generated based on the previous state, similar to a Markov process. This algorithm implements a technique which is known as importance sampling. It attempts to find the states which are most important, i.e. have a configuration as close to optimal as possible. In order to do so it performs n samples where any atom might have its spin flipped in order to see whether this improves the configuration. If it does – meaning the total energy is lowered – the flip is accepted, otherwise it is accepted with a probability proportionate to the energy change. This is done to avoid local minima. Because the more important configurations must be found first, one typically starts recording statistics regarding the energy, specific heat and magnetization after some waiting period. This allows the system to "relax" such that measurements are not "noised" by unimportant configurations. Typically, the number of waiting steps is $\frac{n}{10}$.

2 Results

I have implemented the above algorithm for the one-dimensional and two-dimensional case. This implementation is written in Python in conjunction with the NumPy library. Plots were generated using Matplotlib. The source code for the one- and two-dimensional cases can be found in Appendix A and Appendix B, respectively.

The following tables (Table 1 through Table 6) give the results for the one dimensional case. Above each table it denoted of how many atoms the one-dimensional lattice consisted (by N) and how many MMC steps were taken (by the number of samples). The first column of each table denotes the temperature and the second column the inverse of the temperature, β. The third and fourth columns show the energy and specific heat per spin as found by the MMC method. The fifth and sixth columns show the theoretical, optimal values for the lattice at that temperature regarding energy and specific heat. The final column shows the accuracy of the MMC method. This is computed as the average of the accuracy of the findings of the energy and the accuracy of the specific heat.

The first thing to note from all tables is that for all temperatures larger than 0.5 the found value is fairly close to the theoretical value. This is reflected in the accuracy percentage which is for almost all measurements larger than 85%. When the temperature is too low, the Boltzmann factor $e^{-\beta\delta_E}$ is also very low so very few spin flips are accepted in the MMC algorithm. When the temperature increases, the accuracy also increases.

When the number of samples increase, the accuracy of the MMC also increases. This is to be expected as the it has a higher probability of ending up in a better configuration as more trials are performed. More noteworthy, the number of atoms is a direct influence on the accuracy: more atoms means a higher accuracy. This is because the theoretical values only hold for an infinite system. More atoms means a closer approximation of this infinite system, and can therefore result in a higher accuracy. It is then no surprise that the best results among the trials is one with 1000 atoms and 10000 MMC steps. This system achieves almost 100% accuracy when compared to the theoretical values.

For sufficiently low temperatures, the system exhibits magnetic properties even in absence of an external magnetic field. That is, neighboring spins interact on each other in such a way that they produce a magnetic moment. This is true for the one-dimensional Ising model. It is then interesting to see for which temperature the system undergoes a phase transition from an ordered, spontaneously magnetized system to a disordered system that has no magnetic properties – unless activated by an external magnetic field. However, Ising has proven that the one-dimensional system does not undergo a phase transition[2] and we see no such transition in our results.

2.1 1D results

Table 1: N = 10, 1000 samples

T	β	U_{MC}	C_{MC}	U_{theory}	C_{theory}	Acc
4.00E+00	2.50E-01	-2.09E-01	5.44E-02	-2.45E-01	5.88E-02	87.51%
3.80E+00	2.63E-01	-2.30E-01	5.76E-02	-2.57E-01	6.47E-02	88.02%
3.60E+00	2.78E-01	-2.35E-01	6.49E-02	-2.71E-01	7.15E-02	87.17%
3.40E+00	2.94E-01	-2.46E-01	7.03E-02	-2.86E-01	7.94E-02	85.37%
3.20E+00	3.12E-01	-2.71E-01	8.58E-02	-3.03E-01	8.87E-02	92.54%
3.00E+00	3.33E-01	-2.87E-01	9.19E-02	-3.22E-01	9.96E-02	89.74%
2.80E+00	3.57E-01	-3.01E-01	1.00E-01	-3.43E-01	1.13E-01	86.97%
2.60E+00	3.85E-01	-3.23E-01	1.11E-01	-3.67E-01	1.28E-01	85.86%
2.40E+00	4.17E-01	-3.56E-01	1.42E-01	-3.94E-01	1.47E-01	93.18%
2.20E+00	4.55E-01	-3.64E-01	1.58E-01	-4.26E-01	1.69E-01	87.85%
2.00E+00	5.00E-01	-4.14E-01	1.77E-01	-4.62E-01	1.97E-01	88.77%
1.80E+00	5.56E-01	-4.54E-01	2.05E-01	-5.05E-01	2.30E-01	88.33%
1.60E+00	6.25E-01	-4.89E-01	2.66E-01	-5.55E-01	2.70E-01	92.55%
1.40E+00	7.14E-01	-5.61E-01	3.11E-01	-6.13E-01	3.18E-01	94.14%
1.20E+00	8.33E-01	-5.98E-01	3.48E-01	-6.82E-01	3.71E-01	89.56%
1.00E+00	1.00E+00	-6.47E-01	4.65E-01	-7.62E-01	4.20E-01	86.26%
8.00E-01	1.25E+00	-7.76E-01	3.58E-01	-8.48E-01	4.38E-01	84.19%
6.00E-01	1.67E+00	-8.34E-01	3.27E-01	-9.31E-01	3.70E-01	87.72%
4.00E-01	2.50E+00	-8.84E-01	2.42E-01	-9.87E-01	1.66E-01	78.56%
2.00E-01	5.00E+00	-8.99E-01	8.99E-02	-1.00E+00	4.54E-03	46.94%

Table 2: N = 10, 10000 samples

T	β	U_{MC}	C_{MC}	U_{theory}	C_{theory}	Acc
4.00E+00	2.50E-01	-2.21E-01	5.31E-02	-2.45E-01	5.88E-02	89.34%
3.80E+00	2.63E-01	-2.38E-01	5.88E-02	-2.57E-01	6.47E-02	90.93%
3.60E+00	2.78E-01	-2.45E-01	6.46E-02	-2.71E-01	7.15E-02	89.47%
3.40E+00	2.94E-01	-2.60E-01	7.23E-02	-2.86E-01	7.94E-02	89.99%
3.20E+00	3.12E-01	-2.75E-01	8.07E-02	-3.03E-01	8.87E-02	89.99%
3.00E+00	3.33E-01	-2.87E-01	8.91E-02	-3.22E-01	9.96E-02	88.07%
2.80E+00	3.57E-01	-3.07E-01	1.03E-01	-3.43E-01	1.13E-01	89.53%
2.60E+00	3.85E-01	-3.33E-01	1.17E-01	-3.67E-01	1.28E-01	90.08%
2.40E+00	4.17E-01	-3.52E-01	1.32E-01	-3.94E-01	1.47E-01	88.49%
2.20E+00	4.55E-01	-3.79E-01	1.53E-01	-4.26E-01	1.69E-01	88.63%
2.00E+00	5.00E-01	-4.14E-01	1.74E-01	-4.62E-01	1.97E-01	87.74%
1.80E+00	5.56E-01	-4.52E-01	2.10E-01	-5.05E-01	2.30E-01	89.39%
1.60E+00	6.25E-01	-4.96E-01	2.35E-01	-5.55E-01	2.70E-01	86.64%
1.40E+00	7.14E-01	-5.51E-01	2.77E-01	-6.13E-01	3.18E-01	86.91%
1.20E+00	8.33E-01	-6.11E-01	3.43E-01	-6.82E-01	3.71E-01	90.11%
1.00E+00	1.00E+00	-6.78E-01	3.86E-01	-7.62E-01	4.20E-01	89.36%
8.00E-01	1.25E+00	-7.69E-01	3.93E-01	-8.48E-01	4.38E-01	89.14%
6.00E-01	1.67E+00	-8.41E-01	3.23E-01	-9.31E-01	3.70E-01	87.41%
4.00E-01	2.50E+00	-8.87E-01	1.78E-01	-9.87E-01	1.66E-01	90.92%
2.00E-01	5.00E+00	-9.00E-01	2.50E-02	-1.00E+00	4.54E-03	53.52%

Table 3: N = 100, 1000 samples

T	β	U_{MC}	C_{MC}	U_{theory}	C_{theory}	Acc
4.00E+00	2.50E-01	-2.40E-01	5.76E-02	-2.45E-01	5.88E-02	97.97%
3.80E+00	2.63E-01	-2.57E-01	6.27E-02	-2.57E-01	6.47E-02	98.39%
3.60E+00	2.78E-01	-2.66E-01	6.78E-02	-2.71E-01	7.15E-02	96.44%
3.40E+00	2.94E-01	-2.84E-01	8.30E-02	-2.86E-01	7.94E-02	97.46%
3.20E+00	3.12E-01	-3.01E-01	8.64E-02	-3.03E-01	8.87E-02	98.31%
3.00E+00	3.33E-01	-3.19E-01	9.64E-02	-3.22E-01	9.96E-02	97.92%
2.80E+00	3.57E-01	-3.38E-01	1.10E-01	-3.43E-01	1.13E-01	97.96%
2.60E+00	3.85E-01	-3.66E-01	1.27E-01	-3.67E-01	1.28E-01	99.31%
2.40E+00	4.17E-01	-3.90E-01	1.45E-01	-3.94E-01	1.47E-01	98.75%
2.20E+00	4.55E-01	-4.20E-01	1.74E-01	-4.26E-01	1.69E-01	98.05%
2.00E+00	5.00E-01	-4.56E-01	2.03E-01	-4.62E-01	1.97E-01	97.69%
1.80E+00	5.56E-01	-5.04E-01	2.49E-01	-5.05E-01	2.30E-01	96.21%
1.60E+00	6.25E-01	-5.55E-01	2.76E-01	-5.55E-01	2.70E-01	98.97%
1.40E+00	7.14E-01	-6.05E-01	3.22E-01	-6.13E-01	3.18E-01	98.70%
1.20E+00	8.33E-01	-6.76E-01	3.51E-01	-6.82E-01	3.71E-01	96.60%
1.00E+00	1.00E+00	-7.53E-01	4.66E-01	-7.62E-01	4.20E-01	94.48%
8.00E-01	1.25E+00	-8.46E-01	4.96E-01	-8.48E-01	4.38E-01	94.03%
6.00E-01	1.67E+00	-9.01E-01	4.56E-01	-9.31E-01	3.70E-01	88.85%
4.00E-01	2.50E+00	-9.84E-01	7.70E-01	-9.87E-01	1.66E-01	60.66%
2.00E-01	5.00E+00	-9.78E-01	2.79E+00	-1.00E+00	4.54E-03	48.95%

Table 4: N = 100, 10000 samples

T	β	U_{MC}	C_{MC}	U_{theory}	C_{theory}	Acc
4.00E+00	2.50E-01	-2.43E-01	5.74E-02	-2.45E-01	5.88E-02	98.38%
3.80E+00	2.63E-01	-2.55E-01	6.53E-02	-2.57E-01	6.47E-02	99.16%
3.60E+00	2.78E-01	-2.68E-01	7.10E-02	-2.71E-01	7.15E-02	99.18%
3.40E+00	2.94E-01	-2.84E-01	7.88E-02	-2.86E-01	7.94E-02	99.23%
3.20E+00	3.12E-01	-3.01E-01	8.76E-02	-3.03E-01	8.87E-02	99.04%
3.00E+00	3.33E-01	-3.18E-01	1.00E-01	-3.22E-01	9.96E-02	99.20%
2.80E+00	3.57E-01	-3.39E-01	1.11E-01	-3.43E-01	1.13E-01	98.75%
2.60E+00	3.85E-01	-3.63E-01	1.24E-01	-3.67E-01	1.28E-01	98.02%
2.40E+00	4.17E-01	-3.89E-01	1.44E-01	-3.94E-01	1.47E-01	98.34%
2.20E+00	4.55E-01	-4.20E-01	1.68E-01	-4.26E-01	1.69E-01	98.87%
2.00E+00	5.00E-01	-4.58E-01	1.96E-01	-4.62E-01	1.97E-01	99.37%
1.80E+00	5.56E-01	-5.00E-01	2.36E-01	-5.05E-01	2.30E-01	98.40%
1.60E+00	6.25E-01	-5.49E-01	2.68E-01	-5.55E-01	2.70E-01	99.00%
1.40E+00	7.14E-01	-6.09E-01	3.20E-01	-6.13E-01	3.18E-01	99.41%
1.20E+00	8.33E-01	-6.75E-01	3.55E-01	-6.82E-01	3.71E-01	97.14%
1.00E+00	1.00E+00	-7.54E-01	4.03E-01	-7.62E-01	4.20E-01	97.43%
8.00E-01	1.25E+00	-8.41E-01	4.29E-01	-8.48E-01	4.38E-01	98.47%
6.00E-01	1.67E+00	-9.22E-01	3.46E-01	-9.31E-01	3.70E-01	96.13%
4.00E-01	2.50E+00	-9.72E-01	2.61E-01	-9.87E-01	1.66E-01	81.11%
2.00E-01	5.00E+00	-9.90E-01	2.12E-01	-1.00E+00	4.54E-03	50.57%

Table 5: N = 1000, 1000 samples

T	β	U_{MC}	C_{MC}	U_{theory}	C_{theory}	Acc
4.00E+00	2.50E-01	-2.43E-01	6.53E-02	-2.45E-01	5.88E-02	94.67%
3.80E+00	2.63E-01	-2.54E-01	7.25E-02	-2.57E-01	6.47E-02	94.06%
3.60E+00	2.78E-01	-2.70E-01	7.65E-02	-2.71E-01	7.15E-02	96.55%
3.40E+00	2.94E-01	-2.85E-01	8.97E-02	-2.86E-01	7.94E-02	94.08%
3.20E+00	3.12E-01	-3.03E-01	8.94E-02	-3.03E-01	8.87E-02	99.56%
3.00E+00	3.33E-01	-3.22E-01	1.06E-01	-3.22E-01	9.96E-02	96.77%
2.80E+00	3.57E-01	-3.42E-01	1.31E-01	-3.43E-01	1.13E-01	92.94%
2.60E+00	3.85E-01	-3.66E-01	1.50E-01	-3.67E-01	1.28E-01	92.64%
2.40E+00	4.17E-01	-3.93E-01	1.78E-01	-3.94E-01	1.47E-01	90.93%
2.20E+00	4.55E-01	-4.25E-01	2.09E-01	-4.26E-01	1.69E-01	90.32%
2.00E+00	5.00E-01	-4.64E-01	2.41E-01	-4.62E-01	1.97E-01	90.69%
1.80E+00	5.56E-01	-5.04E-01	3.24E-01	-5.05E-01	2.30E-01	85.49%
1.60E+00	6.25E-01	-5.54E-01	4.00E-01	-5.55E-01	2.70E-01	83.75%
1.40E+00	7.14E-01	-6.11E-01	4.62E-01	-6.13E-01	3.18E-01	84.25%
1.20E+00	8.33E-01	-6.84E-01	6.94E-01	-6.82E-01	3.71E-01	76.66%
1.00E+00	1.00E+00	-7.61E-01	9.75E-01	-7.62E-01	4.20E-01	71.52%
8.00E-01	1.25E+00	-8.47E-01	1.33E+00	-8.48E-01	4.38E-01	66.32%
6.00E-01	1.67E+00	-9.33E-01	2.76E+00	-9.31E-01	3.70E-01	56.61%
4.00E-01	2.50E+00	-9.72E-01	7.13E+00	-9.87E-01	1.66E-01	50.40%
2.00E-01	5.00E+00	-9.69E-01	2.41E+01	-1.00E+00	4.54E-03	48.39%

Table 6: N = 1000, 10000 samples

T	β	U_{MC}	C_{MC}	U_{theory}	C_{theory}	Acc
4.00E+00	2.50E-01	-2.44E-01	5.89E-02	-2.45E-01	5.88E-02	99.75%
3.80E+00	2.63E-01	-2.57E-01	6.56E-02	-2.57E-01	6.47E-02	99.14%
3.60E+00	2.78E-01	-2.70E-01	7.21E-02	-2.71E-01	7.15E-02	99.48%
3.40E+00	2.94E-01	-2.85E-01	8.06E-02	-2.86E-01	7.94E-02	99.09%
3.20E+00	3.12E-01	-3.02E-01	9.08E-02	-3.03E-01	8.87E-02	98.79%
3.00E+00	3.33E-01	-3.21E-01	1.01E-01	-3.22E-01	9.96E-02	99.10%
2.80E+00	3.57E-01	-3.43E-01	1.12E-01	-3.43E-01	1.13E-01	99.83%
2.60E+00	3.85E-01	-3.67E-01	1.32E-01	-3.67E-01	1.28E-01	98.50%
2.40E+00	4.17E-01	-3.94E-01	1.47E-01	-3.94E-01	1.47E-01	99.98%
2.20E+00	4.55E-01	-4.25E-01	1.72E-01	-4.26E-01	1.69E-01	99.21%
2.00E+00	5.00E-01	-4.62E-01	2.01E-01	-4.62E-01	1.97E-01	98.91%
1.80E+00	5.56E-01	-5.04E-01	2.39E-01	-5.05E-01	2.30E-01	98.10%
1.60E+00	6.25E-01	-5.54E-01	2.85E-01	-5.55E-01	2.70E-01	97.48%
1.40E+00	7.14E-01	-6.13E-01	3.35E-01	-6.13E-01	3.18E-01	97.50%
1.20E+00	8.33E-01	-6.81E-01	4.12E-01	-6.82E-01	3.71E-01	94.97%
1.00E+00	1.00E+00	-7.61E-01	4.91E-01	-7.62E-01	4.20E-01	92.73%
8.00E-01	1.25E+00	-8.47E-01	5.33E-01	-8.48E-01	4.38E-01	91.04%
6.00E-01	1.67E+00	-9.30E-01	5.62E-01	-9.31E-01	3.70E-01	82.85%
4.00E-01	2.50E+00	-9.85E-01	7.33E-01	-9.87E-01	1.66E-01	61.27%
2.00E-01	5.00E+00	-9.94E-01	2.33E+00	-1.00E+00	4.54E-03	49.79%

2.2 2D results

The two-dimensional case differs on several points from the one-dimensional case. First, elements are now placed on a two-dimensional lattice instead of on a line. This means that particles can not only interact with their left and right neighbor, but also with those above and below it. This changes the definition of the energy of the system to:

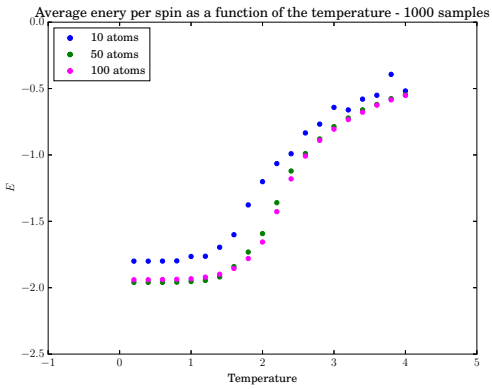
$$E = -\sum_{i=1}^{N-1}\sum_{j=1}^{N} S_{i,j}S_{i+1,j} - \sum_{i=1}^{N}\sum_{j=1}^{N-1} S_{i,j}S_{i,j+1}.$$

Contrary to the one-dimensional case, a phase transition *is* possible in the two-dimensional case as proven by Onsager [3], which was also attempted to observe here. To do this, there are again statistics collected during the MMC computation to determine the average energy per spin U/N^2, specific heat per spin C/N^2 and the average magnetization per spin $\langle M\rangle/N^2$. The magnetization of a configuration is determined by $M = \sum_i \sigma_i$. The average magnetization is thus $\langle M\rangle = \sum_{m\in M} m/|M|$.

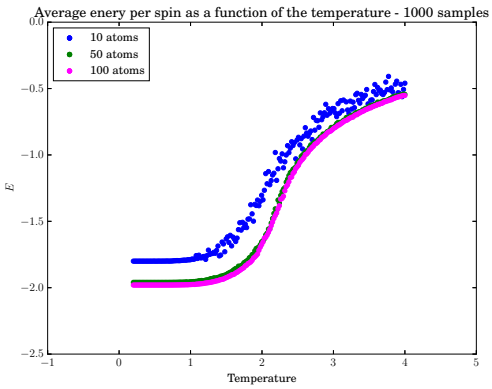
In Figures 1 and 2 show each 6 scatter plots. The first row of each figure shows the average energy per spin the system exhibits for a certain temperature. The second row of each figure shows the specific heat per spin for a certain temperature. The third row of each figure shows the average magnetization per spin for each temperature. The left column of each figure shows the plots for 20 different temperatures whereas the right column shows the same plot for 200 different temperatures.

As we can see from the plots, a phase transition is occurring here. This is visible in all the scatter plots. In the energy plots (Figures 1a, 1b, 2a and 2b), we can see that the energy begins to increase starting from a temperature of about 1.5. This increase reaches its apex between $2.2 \leq t \leq 2.5$ after which the energy increase begins to slow down. This is also confirmed in the specific heat plots (Figures 1c, 1d, 2c and 2d). As the specific heat is the derivative of the energy w.r.t. temperature, we can interpret the peak at a temperature of about 2.3 as the temperature at which the speed of energy increase reaches a maximum. This is thus also where the phase transition happens. Not coincidentally, the critical temperature $T_c = \frac{2}{\ln(1+\sqrt{2})} \approx 2.269$. For the systems with 10×10 atoms, this peak is further away from T_c because this system is too small and the equations describing the infinite system do not reasonably describe the small, finite system.

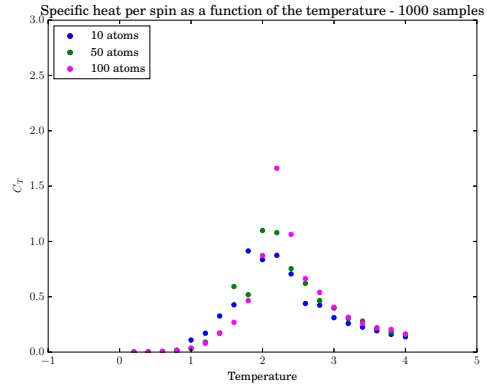
In the magnetization plots (Figures 1e, 1f, 2e and 2f) we can also see a clear phase transition. For temperatures below about 2.3, there is very often a non-zero average magnetization. This means that spontaneous magnetization is occurring. However, beyond the critical temperature, spontaneous magnetization is not anymore occuring, because order disappears in the system due to a too high temperature. There are too many spins flipping, causing spontaneous magnetization to disappear.



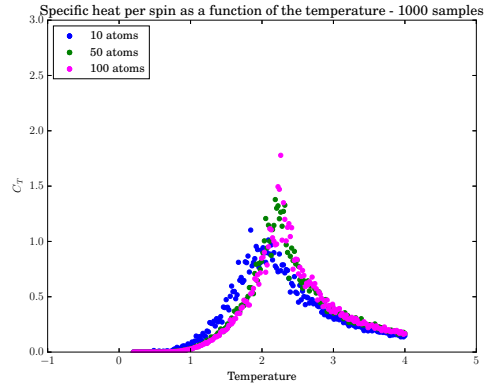
(a)



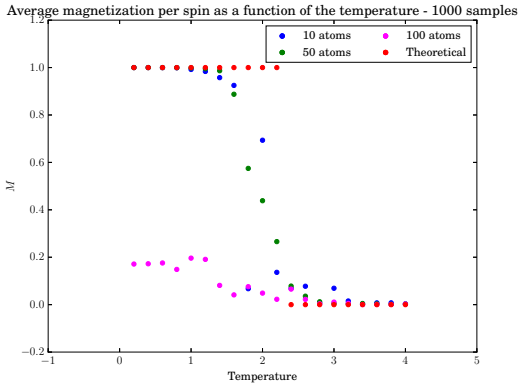
(b)



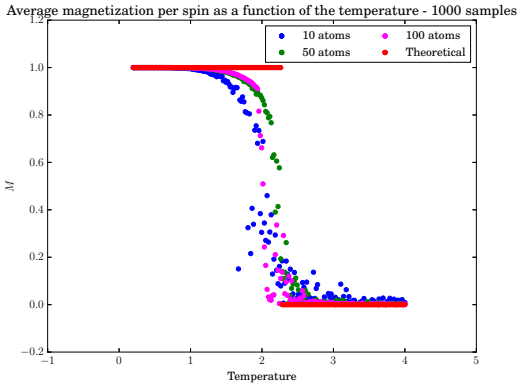
(c)



(d)



(e)



(f)

Figure 1: Scatter plots for 1000 MMC steps. Figures a,c and e show the average energy, specific heat and average magnetization for 20 different temperatures, respectively. Figures b,d and f show the same for 200 different temperatures.

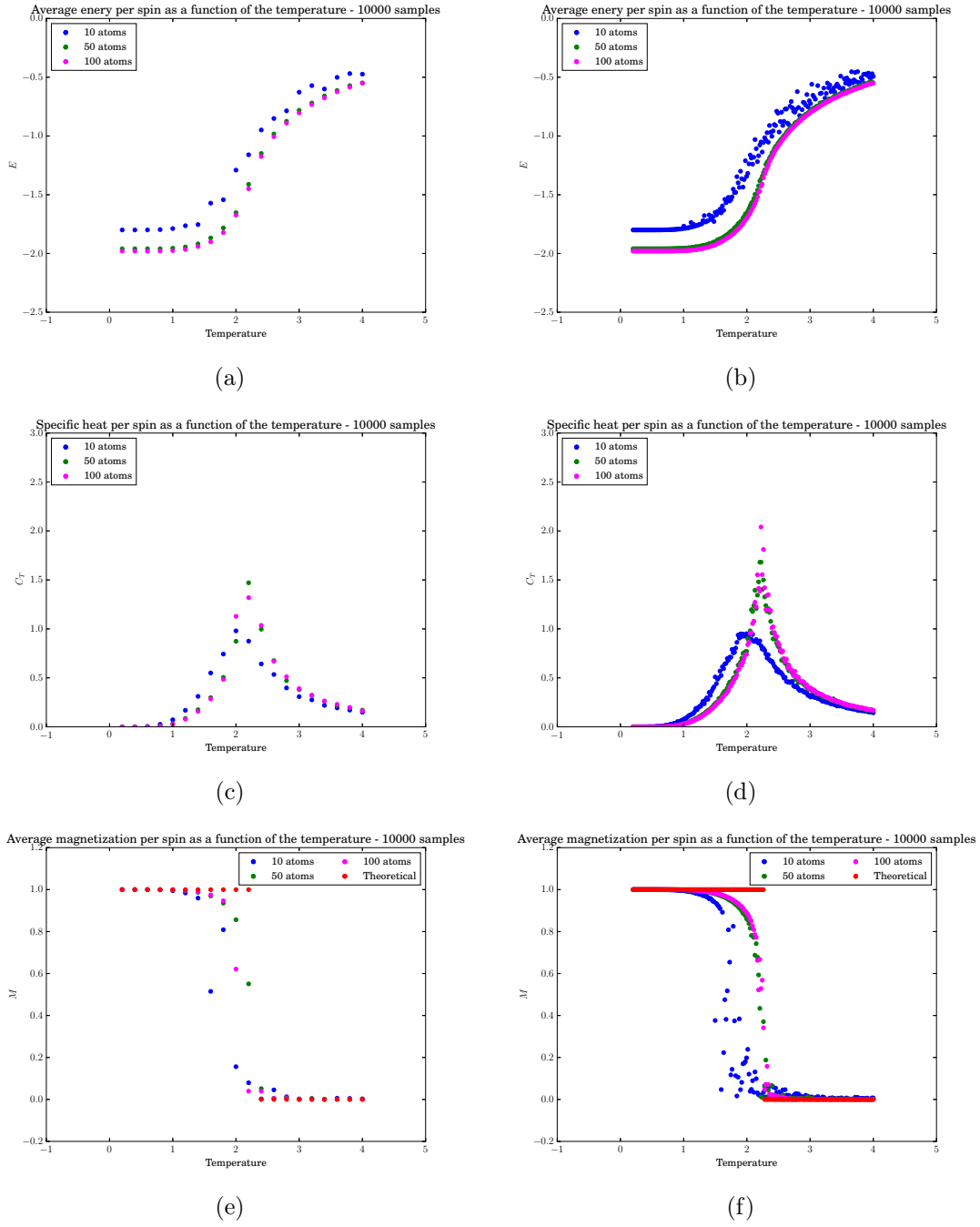


Figure 2: Scatter plots for 10000 MMC steps. Figures a,c and e show the average energy, specific heat and average magnetization for 20 different temperatures, respectively. Figures b,d and f show the same for 200 different temperatures.

Conclusion

The MMC method succeeds in finding (near)-optimal configurations of the Ising model with respect to the total energy of the system. The results obtained are consistent with the mathematical results provided by Ising and Onsager, as there were no phase transitions found in the one-dimensional case. However, there was a phase transition observed from the ordered, spontaneously magnetized case to the unordered situation without spontaneous magnetization. This happens near the theoretical critical temperature $T_c = \frac{2}{\ln(1+\sqrt{2})} \approx 2.269$.

A 1D code

```
#!/usr/bin/env python
import numpy as np
from tabulate import tabulate
import random


def config_energy(S):
    # efficient energy calculation using dot products
    return -np.dot(S[:-1], S[1:])


def pick_config(N):
    options = [-1, 1]
    return np.random.choice(options, N)


def determine_average_energy(beta, energy, n):
    avg_energy = np.mean(energy) / n
    theo_energy = -np.tanh(beta)
    acc = 1 - np.abs((np.abs(avg_energy - theo_energy) / avg_energy))
    return (avg_energy, theo_energy, acc)


def determine_specific_heat(beta, energy, n):
    spec_heat = (np.var(energy) / n) * (beta ** 2)
    theo_heat = (beta / np.cosh(beta)) ** 2
    acc = 1 - np.abs((np.abs(spec_heat - theo_heat) / spec_heat))
    return (spec_heat, theo_heat, acc)
```

```
def find_energy_change(random_idx, config):
    conf_len = len(config)
    fp = 0
    sp = 0
    # no wrap around, so edge cases handled separately
    if random_idx == 0:
        fp = config[0] * config[1] - (-config[0] * config[1])
    elif random_idx == conf_len - 1:
        sp = config[-2] * config[-1] - (-config[-1] * config[-2])
    else:
        fp = config[random_idx] * config[random_idx + 1] - \
            (-config[random_idx] * config[random_idx + 1])
        sp = config[random_idx-1] * config[random_idx] - \
            (-config[random_idx] * config[random_idx - 1])
    return fp + sp


def mmc(beta, n, n_samples, config=None, n_wait=None):
    # start with a random configuration and determine its energy
    if config is None:
        config = pick_config(n)
    energy = config_energy(config)
    energies = np.array([])
    # default waiting time is N / 10
    if n_wait is None:
        n_wait = int(n_samples / 10)
    # Take a number of samples + waiting time (to let it relax)
    for l in range(n_samples + n_wait):
        for z in range(n):
            # pick a random atom
            j = np.random.randint(0, n)
            # Determine difference in energy
            energy_prime = find_energy_change(j, config)
            q = np.exp(-beta * energy_prime)
            r = random.random()
            # Always do it if the energy is lower, else accept it with a
            # probability
            if energy_prime < 0 or q > r:
                # flip its spin
                config[j] = -config[j]
```

```
                energy += energy_prime
        # if the system is somewhat relaxed, store the energy
        if l > n_wait:
            energies = np.append(energies, energy)
    return (energies, config)


def do_simulation(n, sample):
    betas = [1/t for t in np.linspace(0.2, 4, 20)]
    headers = ["T", "beta", "U_MC", "C_MC", "U_theory", "C_theory", "Acc"]
    # here the results are stored. Key is the tuple of number of samples
    # and the number of atoms. Value is an array with statistics in the order
    # of the headers array
    datas = dict()

    # Last column separately added so tabulate doesn't mess up the
    # formatting
    data = np.zeros((len(betas), len(headers) - 1))
    a = np.array([""] * len(betas))
    data = np.c_[data, a]
    config = None
    # reverse betas for aesthetics
    for i, beta in enumerate(betas[::-1]):
        (energy, config) = mmc(beta, n, sample, config)

        (avg_energy, theo_e, acc_e) = determine_average_energy(
            beta, energy, n)
        (spec_heat, theo_c, acc_h) = determine_specific_heat(
            beta, energy, n)
        table_row = [1 / beta, beta, avg_energy, spec_heat,
                     theo_e, theo_c, "{:.2%}".format((acc_h + acc_e) / 2)]
        data[i, :] = table_row

    datas[(sample, n)] = data
    print("Done {0} atoms at {1} samples".format(n, sample))
    for (s, n) in datas.keys():
        print("N = {1}, {0} samples".format(s, n))
        print(
            tabulate(datas[(s, n)], headers=headers, floatfmt='.2E'))
        print("")
```

```
for n in [10, 100, 1000]:
    for sample in [1000, 10000]:
        do_simulation(n, sample)
```

B 2D code

```
#!/usr/bin/python
import numpy as np
import random
import multiprocessing
from math import log, sinh
from itertools import repeat
import sys
import time
import matplotlib.pyplot as plt


def pick_config(N):
    options = [-1, 1]
    return np.random.choice(options, (N, N))


def config_energy(lattice):
    # efficient energy calculation using dot products
    fp = np.dot(lattice[:-1, :].ravel(), lattice[1:, :].ravel())
    sp = np.dot(lattice[:, :-1].ravel(), lattice[:, 1:].ravel())
    return -fp - sp

config = None


def find_change_in_energy(i, j, config):
    m, n = config.shape
    orig, new = 0.0, 0.0
    if i == 0:
        if j == 0:
            # top left corner
            orig = -config[i, j] * config[i + 1, j] - \
                config[i, j] * config[i, j + 1]
            config[i, j] = -config[i, j]
            new = -config[i, j] * config[i + 1, j] - \
```

```
                config[i, j] * config[i, j + 1]
            config[i, j] = -config[i, j]
        elif j == n - 1:
            # top right corner
            orig = -config[i, j] * config[i + 1, j] - \
                config[i, j - 1] * config[i, j]

            config[i, j] = -config[i, j]
            new = -config[i, j] * config[i + 1, j] - \
                config[i, j - 1] * config[i, j]
            config[i, j] = -config[i, j]

        else:
            # just along top strip
            left = config[i, j - 1] * config[i, j]
            right = config[i, j] * config[i, j + 1]
            orig = -config[i, j] * config[i + 1, j] - left - right
            config[i, j] = -config[i, j]
            left = config[i, j - 1] * config[i, j]
            right = config[i, j] * config[i, j + 1]
            new = -config[i, j] * config[i + 1, j] - left - right
            config[i, j] = -config[i, j]
    elif i == n - 1:
        if j == 0:
            # bottom left corner
            orig = -config[i - 1, j] * config[i, j] - \
                config[i, j] * config[i, j + 1]
            config[i, j] = -config[i, j]
            new = -config[i - 1, j] * config[i, j] - \
                config[i, j] * config[i, j + 1]
            config[i, j] = -config[i, j]

        elif j == n - 1:
            # bottom right corner
            orig = -config[i - 1, j] * config[i, j] - \
                config[i, j - 1] * config[i, j]
            config[i, j] = -config[i, j]
            new = -config[i - 1, j] * config[i, j] - \
                config[i, j - 1] * config[i, j]
            config[i, j] = -config[i, j]
        else:
```

```
            # along bottom strip
            left = config[i, j - 1] * config[i, j]
            right = config[i, j] * config[i, j + 1]
            orig = -config[i, j] * config[i - 1, j] - left - right
            config[i, j] = -config[i, j]
            left = config[i, j - 1] * config[i, j]
            right = config[i, j] * config[i, j + 1]
            new = -config[i, j] * config[i - 1, j] - left - right
            config[i, j] = -config[i, j]

    elif j == 0:
        # along left strip, rest handled
        above = config[i - 1, j] * config[i, j]
        below = config[i, j] * config[i + 1, j]
        orig = -config[i, j] * config[i, j + 1] - above - below
        config[i, j] = -config[i, j]
        above = config[i - 1, j] * config[i, j]
        below = config[i, j] * config[i + 1, j]
        new = -config[i, j] * config[i, j + 1] - above - below
        config[i, j] = -config[i, j]

    elif j == n-1:
        # along right strip, rest handled
        above = config[i - 1, j] * config[i, j]
        below = config[i, j] * config[i + 1, j]
        orig = -config[i, j] * config[i, j - 1] - above - below
        config[i, j] = -config[i, j]
        above = config[i - 1, j] * config[i, j]
        below = config[i, j] * config[i + 1, j]
        new = -config[i, j] * config[i, j - 1] - above - below
        config[i, j] = -config[i, j]

    else:
        # all directions available
        above = config[i - 1, j] * config[i, j]
        below = config[i, j] * config[i + 1, j]
        left = config[i, j - 1] * config[i, j]
        right = config[i, j] * config[i, j + 1]
        orig = -left - right - above - below
        config[i, j] = -config[i, j]
        above = config[i - 1, j] * config[i, j]
```

```
            below = config[i, j] * config[i + 1, j]
            left = config[i, j - 1] * config[i, j]
            right = config[i, j] * config[i, j + 1]
            new = -left - right - above - below
            config[i, j] = -config[i, j]

    return new - orig


def mmc(beta, n, n_samples, config=None, n_wait=None):
    # Find the optimal configuration for a temperature using MMC
    # start with a random configuration and determine its energy
    if config is None:
        config = pick_config(n)
    energy = config_energy(config)
    energies = np.array([])
    mags = np.array([])
    # default waiting time is N / 10
    if n_wait is None:
        n_wait = n_samples // 10
    # Take a number of samples + waiting time (to let it relax)
    for l in range(n_samples + n_wait):
        for z in range(n*n):
            # pick a random atom
            j = np.random.randint(0, n)
            i = np.random.randint(0, n)
            # Determine difference in energy
            energy_prime = find_change_in_energy(i, j, config)
            q = np.exp(-beta * energy_prime)
            r = random.random()
            # Always accept if it's better, else accept it with a probability
            if q > r:
                energy += energy_prime
                config[i, j] = -config[i, j]
            # if it is not accepted, revert to old situation
        # if the system is somewhat relaxed, store the energy
        if l > n_wait:
            energies = np.append(energies, energy)
            mags = np.append(mags, np.sum(config.ravel()))
    return (energies, mags, config)
```

```
def compute_lattice(beta_n_nsamp_config):
    # I do not like PEP-3113
    beta, n, nsamp, config = beta_n_nsamp_config
    Energies, Mags, config = mmc(beta, n, nsamp, config)
    energy_per_spin = np.mean(Energies) / (n * n)
    spec_heat_per_spin = (np.var(Energies) * beta ** 2) / (n * n)
    avg_magnetization = np.mean(Mags) / (n * n)
    return (1 / beta, energy_per_spin, spec_heat_per_spin, avg_magnetization, config)


if len(sys.argv) != 3:
    print("Usage: mmc_2d <num points> <num samples>")
    print("     num points:  number of temperatures between 0.2 and 4")
    print("                  Note: by taking this value as 20 you will get")
    print("                        the interval the assignment requires")
    print("     num samples: number MMC steps to perform per temperature")
    sys.exit(-1)


def calc(result_queue, betas, n, num_samples, config=None):
    ts = []
    energies_ps = []
    spec_heats_ps = []
    mags_ps = []

    for i, beta in enumerate(betas[::-1]):
        tic = time.time()
        t, energy_ps, spec_heat_ps, mag_ps, config = compute_lattice(
            (beta, n, num_samples, config))
        elapsed = time.time() - tic
        print("Done {0}, {1}% done. This took {2} seconds".format(
            beta, (((i + 1) / float(len(betas))) * 100.0), elapsed))
        ts.append(t)
        energies_ps.append(energy_ps)
        spec_heats_ps.append(spec_heat_ps)
        mags_ps.append(mag_ps)

    result_queue.put((ts, energies_ps, spec_heats_ps, mags_ps))

num_points = int(sys.argv[1])
```

```
num_samples = int(sys.argv[2])
betas = [1/t for t in np.linspace(0.2, 4, num_points)]
# # Compute all statistics for 10, 50, and 100 atoms respectively in parallel
# # but sequentially.
result_queue = multiprocessing.Queue()
jobs = [multiprocessing.Process(
    target=calc, args=(result_queue, betas, n, num_samples)) for n in [10,
                                                                        50, 100]]

for job in jobs:
    job.start()
for job in jobs:
    job.join()
results = [result_queue.get() for j in jobs]

ts_10, energies_ps_10, spec_heats_ps_10, mags_ps_10 = results[0]
ts_50, energies_ps_50, spec_heats_ps_50, mags_ps_50 = results[1]
ts_100, energies_ps_100, spec_heats_ps_100, mags_ps_100 = results[2]

plt.rc('text', usetex=True)
plt.rc('font', family='serif')
crit_temp = 2 / (log(1 + (2 ** 0.5)))

theo_mag = [(1 - sinh(2 / t) ** -4) ** (1/8) if t <
            crit_temp else 0 for t in ts_10]

# Scatter plot of the energy
e_10 = plt.scatter(ts_10, energies_ps_10, color='blue')
e_50 = plt.scatter(ts_50, energies_ps_50, color='green')
e_100 = plt.scatter(
    ts_100, energies_ps_100, color='magenta')
axes = plt.gca()
energy_title = "Average enery per spin as a function of the " + \
    "temperature - {0} samples".format(num_samples)
plt.title(energy_title)
axes.set_xlabel("Temperature")
axes.set_ylabel("$E$")

plt.legend((e_10, e_50, e_100),
           ('10 atoms', '50 atoms', '100 atoms'),
           scatterpoints=1,
```

```
            loc='upper left',
            ncol=1,
            fontsize=12)

energy_filename = 'energies-{0}-{1}.eps'.format(num_samples, num_points)
plt.savefig(energy_filename, format='eps', dpi=1000)

# Scatter plot of the specific heat
plt.figure()
c_10 = plt.scatter(ts_10, spec_heats_ps_10, color='blue')
c_50 = plt.scatter(
    ts_50, spec_heats_ps_50, color='green')
c_100 = plt.scatter(
    ts_100, spec_heats_ps_100, color='magenta')
plt.legend((c_10, c_50, c_100),
            ('10 atoms', '50 atoms', '100 atoms'),
            scatterpoints=1,
            loc='upper left',
            ncol=1,
            fontsize=12)

axes = plt.gca()
heat_title = "Specific heat per spin as a function of the " + \
    "temperature - {} samples".format(num_samples)
plt.title(heat_title)
axes.set_xlabel("Temperature")
axes.set_ylabel("$C_T$")
axes.set_ylim([0, 3])
heats_filename = 'heats-{0}-{1}.eps'.format(num_samples, num_points)
plt.savefig(heats_filename, format='eps', dpi=1000)

# Scatter plot of the magnetiztion per spin.
# Note it plots the absolute value
plt.figure()
m101000 = plt.scatter(
    ts_10, np.abs(mags_ps_10), color='blue')
m501000 = plt.scatter(
    ts_50, np.abs(mags_ps_50), color='green')
m1001000 = plt.scatter(
    ts_100, np.abs(mags_ps_100), color='magenta')
mtheo = plt.scatter(ts_10, theo_mag, color='red')
```

```
plt.legend((m101000, m501000, m1001000, mtheo),
           ('10 atoms', '50 atoms', '100 atoms', 'Theoretical'),
           scatterpoints=1,
           loc='upper right',
           ncol=2,
           fontsize=12)

axes = plt.gca()
mag_title = "Average magnetization per spin as a function of the " + \
    "temperature - {0} samples".format(num_samples)
plt.title(mag_title)

axes.set_xlabel("Temperature")
axes.set_ylabel("$M$")
mags_filename = 'magnetizations-{0}-{1}.eps'.format(
    num_samples, num_points)
plt.savefig(mags_filename, format='eps', dpi=1000)
plt.show()
```

References

1. Gemma De las Cuevas and Toby S Cubitt. Simple universal models capture all spin physics. *arXiv preprint arXiv:1406.5955*, 2014.
2. Ernst Ising. Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik A Hadrons and Nuclei*, 31(1):253–258, 1925.
3. Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Physical Review*, 65(3-4):117, 1944.