UNIVERSITY OF CAMBRIDGE

Part II Computational Physics Project

Computational Monte Carlo Investigation of the 2D the Ising Model

Anonymised Name
Candidate Number:

Supervisor: Dr. D. Buscher

Department of Physics
University of Cambridge
2018

Abstract

A computational study was made into the behaviour of a two-dimensional lattice of spin sites obeying the spin statistics set out by the Ising model. The study utilised these spin statistics and finite-scaling arguments to verify Onsanger's result for the critical temperature of the second order phase transition for such a system, obtaining $T_C(N = \infty) = 2.23 + 0.04 - 0.05 J_{spin}/k_B$. A series of other results, predicted by universality of phase transitions, were verified qualitatively: namely, investigating the behaviour arising from critical exponents such as divergence of lattice properties at the critical temperature for zero external magnetic field. Finally, the change in the system's behaviour upon application of an external magnetic field was investigated. Amongst other expected behaviours, it was observed that the phase transition became continuous for $B \neq 0$, after the symmetry of the system was broken.

Key Words:

Computational Physics, 2D Ising Model, Critical Temperature, Universality, Critical Exponent, Finite Scaling

Word count

2858 - excluding summary, code listing, figures, references and appendices.

Contents

1 Introduction

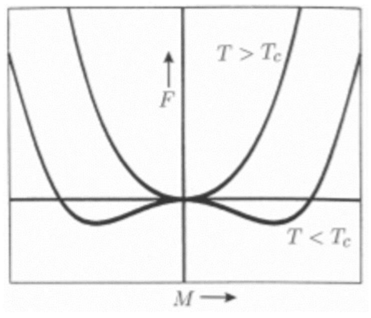
1.1 Introduction to the problem

The Hamiltonian for a lattice represented by the Ising model for spin interactions depends on the sum of spin sites' interactions with their nearest neighbours and an external field:

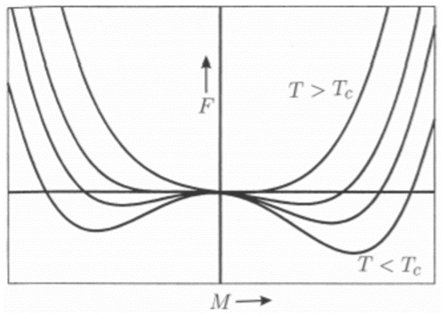
$$E = -J\sum_{i,j}\sigma_i\sigma_j - m_0 B\sum_{i=1}^{N^2}\sigma_i \tag{1}$$

for spin interaction energy J, magnetic moment m_0, external field B and site spin $\sigma = \pm 1$.

Clearly, for $B = 0$, we expect alignment of spins to be the most energetically favourable arrangement, thus for this to be spontaneous. At higher temperatures, however, this favourability becomes insignificant compared to thermal energy and the net spin is expected to reach zero. Figure 1 demonstrates this transition between the two phases in terms of the magnetisations of the free energy minima for various temperatures, with and without a magnetic field, as described by Landau theory.



(a) $B = 0$ - symmetric



(b) $B \neq 0$ - asymmetric

Figure 1: The free energy above and below some critical temperature per net magnetisation, without (a) and with (b) an externally applied magnetic field. Notice the asymmetry grows and the aligned state is favoured for higher field and lower temperature, in figure (b).

The problem of relating net lattice energy and magnetisation with varying magnetic field and temperature has been solved analytically in 1D [1] and 2D [2]. In this experiment we intend to verify Onsanger's 2D result, amongst other theories relating to fluctuations of the system and overall behaviour, with computational Monte Carlo methods.

Key predictions of the phase transition theory, to be investigated in the remainder of the report, are as follows:

- For $B \to 0$, a sharp, second order phase transition in magnetisation is observed when varying temperature. The system transitions from a fully aligned phase to zero net magnetisation at a temperature proportional to J.
- For $B \neq 0$, a smooth variation of the magnetisation with temperature is predicted (thus no sharp phase transition).

1.2 Universality

Furthermore, second order (or continuous) phase transitions predict universality of behaviour around a critical point for many systems. Universality classes (e.g. the Ising model's class) are determined by: the dimensionality of the space (two); the number of components in its order parameter (one- magnetisation); and whether the interaction is short or long ranged (short). If these are all the same for any two systems, it implies the same values of the critical exponents found in table 1.

Quantity	$\sim$	$\lvert T-T_C\rvert^c$
Magnetisation	$M \sim$	$\lvert T-T_C\rvert^{\beta}$
Mag. Susceptibility	$\chi_M \sim$	$\lvert T-T_C\rvert^{-\gamma}$
Heat capacity	$C \sim$	$\lvert T-T_C\rvert^{-\alpha}$
Correlation length	$\tau_e \sim$	$\lvert T-T_C\rvert^{-\mu}$

Table 1: The predictions of universality in second order phase transitions.

In the remainder of this investigation, we will examine whether the quantities found in table 1 do indeed vary as suggested for the 2D Ising model, as well as the predictions of the two bullet-points found in section 1.1.

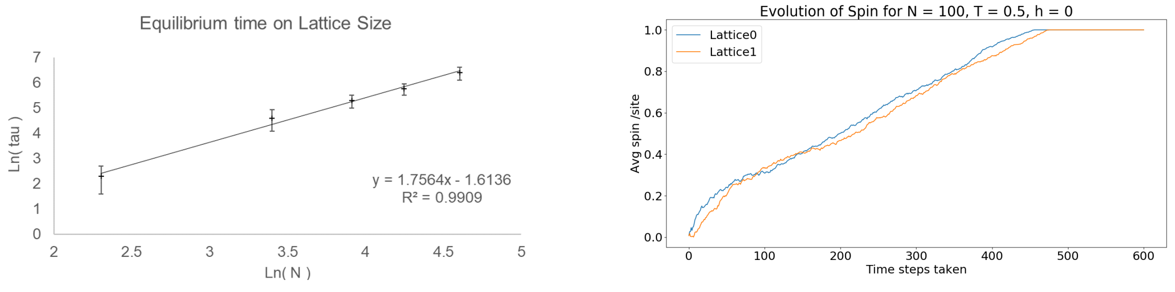
2 Problem set-up and computational choices

The code listing in section B gives an exact implementation of the accurate storage and progression of a finite 2D lattice, however key features and physical considerations in the code will be mentioned in this section.

2.1 Lattice initialisation and equilibrium

The 2D lattice object was initialised as an NxN array of spin-site objects. Section 1 predicts a phase transition between low and high temperature regimes, which called for careful handling of the system's boundary conditions and initialisation to an equilibrium state before experimentation.

A rough investigation into the number of steps required to reach equilibrium, found to be dependent on lattice size, is highlighted in figure 2. Two independently seeded, initially random lattices were run at `low` temperature until they came to equilibrium (i.e. fully aligned). Runs resulting in metastable states (appendix A.1) were ignored in this investigation, and are accounted for in section 2.2.



(a) Overall relation (b) Example data

Figure 2: An interesting (albeit rough) visualisation of the relation between the lattice size N and number of whole-lattice sweeps required to reach equilibrium. Note, runs that resulted in a metastable state (see appendix A.1) are omitted from the figure.

Secondly, based on other studies, periodic boundary conditions were considered to give the most accurate results for experiments, particularly given a general focus on lattice sizes of 70x70 or smaller [3].

2.2 Routines for progression of the lattice

The Metropolis algorithm (appendix A.2) was used for a general progression of the lattice. To satisfy the detailed balance condition (appendix A.4), each time step visited each site in the lattice once and only once in a randomised order. A heatmap visualisation of a typical progression can be viewed in appendix A.3.

Data was to be collected at a range of temperatures, while avoiding metastable states (appendix A.1); each temperature must be visited by a lattice in its global minimum energy state consistently. To ensure this, lattices were generated in an initially aligned state at low temperature[1] and progressed to higher temperatures incrementally. *The previous temperature's final state was used as the following temperature's initial state*, with a sufficient number of whole-lattice sweeps made after each temperature progression to ensure equilibrium before collecting data (the numbers obtained in section 2.1 were deemed to be safe for this purpose).

Finally, data was collected over a fixed number of sweeps larger than the correlation time of the lattice. This consideration will be addressed quantitatively in section 3.1.2.

2.3 Scaling of the Problem

Throughout the computation, all constants were set to unity. Specifically: exchange energy J, electron magnetic moment m_0 and Boltzmann constant k_B (consider equation 1).

Thus, all temperatures in this report are implicitly in scaling-units of J/k_B and magnetic fields in units of J/m_0, thus energies are quoted in units of spin exchange energy J. The result is that behaviour of interactions due to temperature and magnetic field changes can be examined with quantities around the order of unity, which is particularly insightful given the similar values of k_B and m_0.

[1]This is the equilibrium state for low temperatures, as demonstrated in figure 2.

2.4 Performance and efficiency

Given the Metropolis algorithm's quadratic scaling of time complexity with lattice size, a range of optimisations were made to each step in the progression algorithm for satisfactory running time. These optimisations (in order of speed-up) are listed in this section.

Random number generation

Random number generation for Boltzmann probability spin-flipping of sites (appendix A.2) proved to be by far the most expensive part of each progression. A factor two speed-up was achieved by generating an array of random numbers in range $k_BT \ln([0,1))$ only once for each temperature run and comparing directly to ΔE. Array position corresponded to lattice position, and the order of the array was permuted only after each whole-lattice progression.

Correlation by using the same set of random numbers was not an issue, as long as $n_{steps} << N^2$. This could indeed be the case while satisfying equilibrium and correlation times, as roughly shown in figure 2.

Forward computation of ΔE

The Metropolis acceptance rate varies with temperature from 0% to around 70%, thus computing at every step ΔE when checking for a spin flip at each site is an unnecessary expense. Instead, ΔE values were stored by the lattice site objects, only to be recomputed (for the site and its neighbours) in the case that the spin flip is accepted. This led to around a 50% increase in speed for low temperatures to around 20% at regularly used high temperatures (about $5J/k_B$).

NumPy vectorisation

Using NumPy's vectorisation functionality for whole-lattice operations (mostly only used in initialisation), minimal ($\sim$ 15%) speed-ups were accessed. Although implementation is largely the same as running two for-loops over the lattice array, executing this with vectorisation is thought to reduce interpretation time for the language, thus the slightly faster calculation observed.

2.5 Resultant computation times

Here we note some typical CPU times for a single progression of different lattices in equilibrium. Computation was performed on 4 hyperthreads of an i7-6700HQ (2.70GHz) processor with 4GB of 1600MHz DDR3 RAM. Note, however, Python executes on a single core by default (see section 4.4 for further comments on Python).

Lattice Size N	Temp	Flip Percentage	CPU Time (ms)
20	0.5	0%	2
20	5	71%	10
50	0.5	0%	10
50	2.5	34%	40
50	5	70%	60
100	0.5	0%	50
100	5	70%	200

Table 2: Computation times for a single progression of some typical lattices in equilibrium (e.g. aligned at low temp, random at high), averaged over 100s of steps. The parameters for each progression are noted in the table.

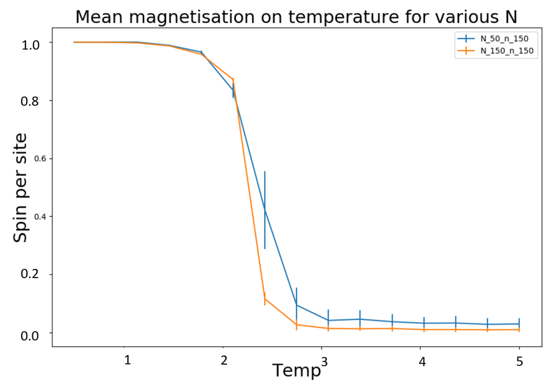
3 Method

Investigations will be undertaken in section 3.1 to understand the behaviour of the system with zero external field, followed by a quantitative study of its properties in section 3.2. Finally, an external field is introduced in section 3.3. Detailed computational explanation will be left in section 2.

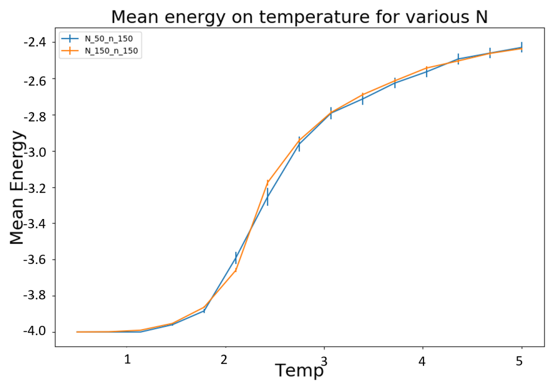
3.1 Initial zero-field investigations

3.1.1 Monitoring evolution of energy and spin for range of temperatures

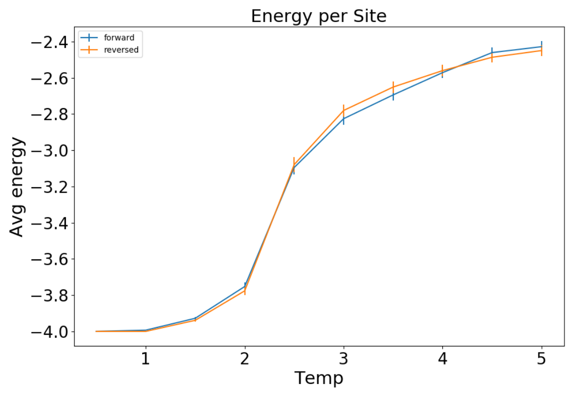
An initial study of the system's energy and spin evolution was made under variation of temperature with the standard progression algorithms (figure 3).



(a) Magnetisation evolution for two N values



(b) Energy evolution for two N values



(c) Reversing progression, N = 50

Figure 3: The evolution of magnetisation and spin for a series of temperatures at two different N values (50 and 150), and positive and negative progression of temperature for N = 50. Observe a dramatic change in system behaviour between temperatures 2 and 3 J/k_B, and the slight smoothing of fluctuations for larger N. Reversal of temperature progression has no effect on results.

From figure 3, we observe a temperature range of interest at $T = 2.49 + 0.16 - 0.29J/k_B$, where the system transitions from fully aligned to net-zero spin phases. The exact position also appears to vary with N, which will be observed more closely in section 3.2.

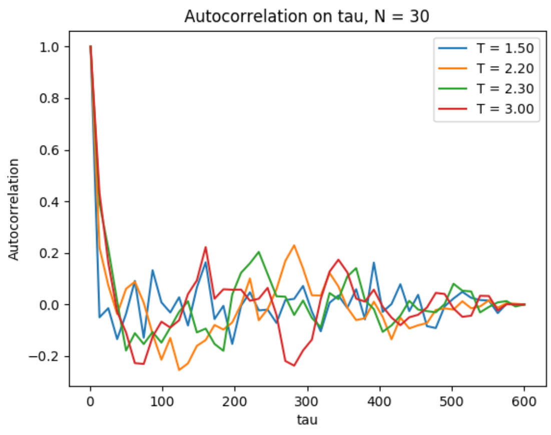
3.1.2 Correlation time

As fluctuations are clearly observed over multiple progressions (e.g. error bars on figure 2), a closer study of these fluctuations was required for quantitative studies going forward. The investigation begins with the definition of the autocorrelation of the magnetisation for a time lag τ:

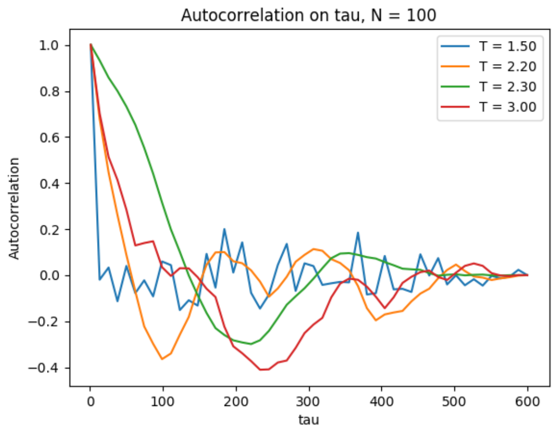
$$a(\tau) = \frac{\langle M'(t)M'(t+\tau)\rangle}{\langle M'(t)^2\rangle}, \tag{2}$$

where $M'(t)$ is the deviation from the mean magnetisation over all time, at a certain time t.

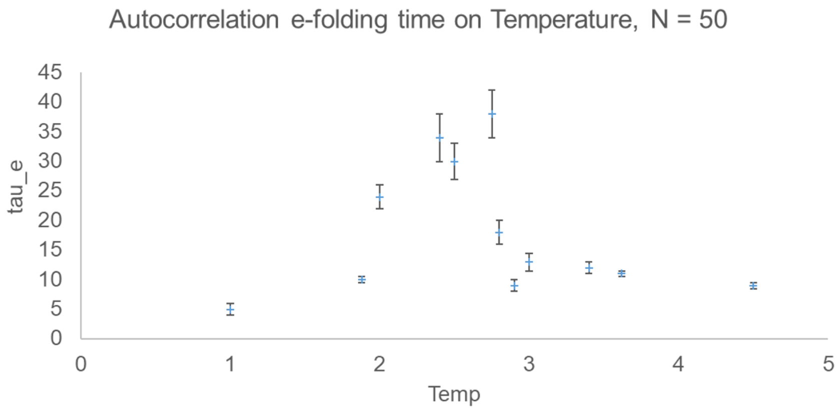
Example data of the results of this study are given in figure 4; e-folding correlation times τ_e are observed to be longest around the critical temperature, and scales with lattice size N. For example, the figure shows that at $T = 2.3J/k_B$, $\tau_{e,max}(N = 30, 100) = 25, 100$.



(a) Autocorrelation for $N = 30$



(b) Autocorrelation for $N = 100$



(c) e-folding times for various temperatures

Figure 4: Example data of the e-folding time of the autocorrelation, τ_e. The e-folding correlation time is observed to increase with lattice size, and diverges as the transition temperature is neared.

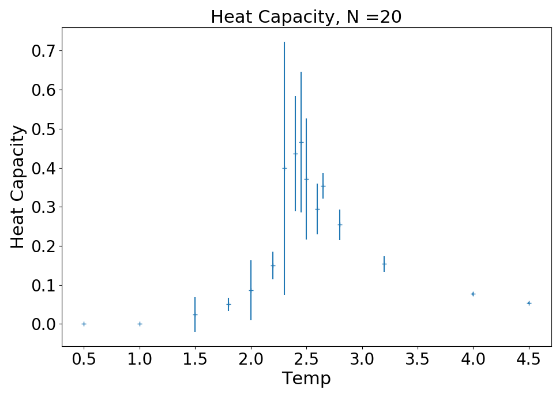
Finally, figure 4(c) shows there is a divergence in the correlation time at the temperature around which we expect to find the phase transition, in line with the predictions in section 1.2.

Together, these results directly informed the remainder of this study, where time to reach equilibrium at each step was taken from section 2.1, and the number of time steps to take data and average over once in equilibrium was sufficiently greater than $\tau_e(N, T)$.

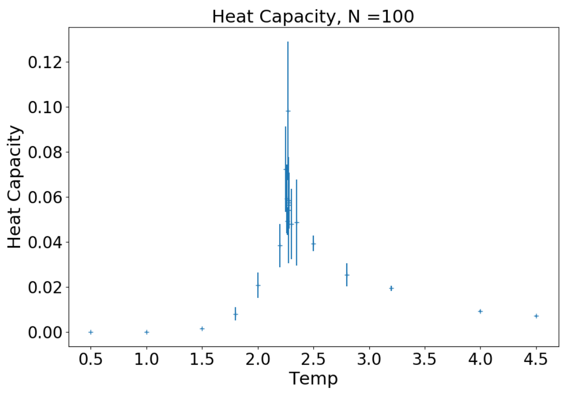
3.2 Finding the critical temperature with heat capacity divergence

Using information obtained thus far, a quantitative study to find the critical temperature T_C was undertaken. The data in figure 5 were generated using the fluctuation-dissipation theorem for lattice energy fluctuations in the equilibrium state [4, 5]:

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



(a) Specific heat capacities for $N = 20$



(b) Specific heat capacities for $N = 100$

Figure 5: Specific heat capacities (units k_B per site) as a function of temperature for two different lattice sizes. The heat capacity is seen to diverge at a certain temperature, which appears to vary with lattice size. Lattices were equilibrated and then averaged over 100 and 600 whole-lattice sweeps respectively. Note: many points used in obtaining the critical temperature are omitted for clarity, in this display.

The key limitation to this study was the size of the statistical error bars, typically obtained over 5 independent runs, thus making it difficult to resolve the position of the divergence in specific heat capacity accurately. A key conclusion from this observation is that the finite lattice does not behave well around T_C, and this algorithm cannot precisely represent the physical system undergoing a transition.

Regardless, some results were carefully obtained to examine the universality properties predicted in section 1.2. Critical temperatures for various lattice sizes can be found in table 3, and will be analysed in further detail in section 4.1. A similar study for the magnetic susceptibility using spin fluctuation did not produce any results to make this study more accurate.

N	T_C	± error
15	2.55	0.05
20	2.48	0.05
30	2.41	0.05
40	2.38	0.05
50	2.37	0.05
60	2.35	0.03
70	2.34	0.03
80	2.33	0.03
100	2.29	0.04

Table 3: Values for the critical temperature obtained for various lattice sizes N, by judging the divergence point in graphs for heat capacities. Error is from uncertainty in the exact determination of the peak position.

3.3 Behaviour with an applied magnetic field

Finally, we investigate the behaviour of the lattice under an applied magnetic field. Section 1.1 makes predictions about this regime that we seek to verify in the following section.

3.3.1 Hysteresis Loops

As predicted by magnetic hysteresis theory, figure 6 shows the progression of magnetisation for forward and reverse external field traversal is asymmetric, thus energy is stored by lattice magnetisation. Figure 1 also demonstrates the metastability in the system being anti-aligned with the magnetic field- there is an energy barrier to be overcome when changing alignment of the system below the critical temperature. Furthermore, the area enclosed by the loop (or energy stored) decreased with increasing temperature. This correctly shows the thermal energy of the system becomes more significant than the thermodynamic disfavour of anti-alignment of a system for high temperature $k_BT >> m_0B$.

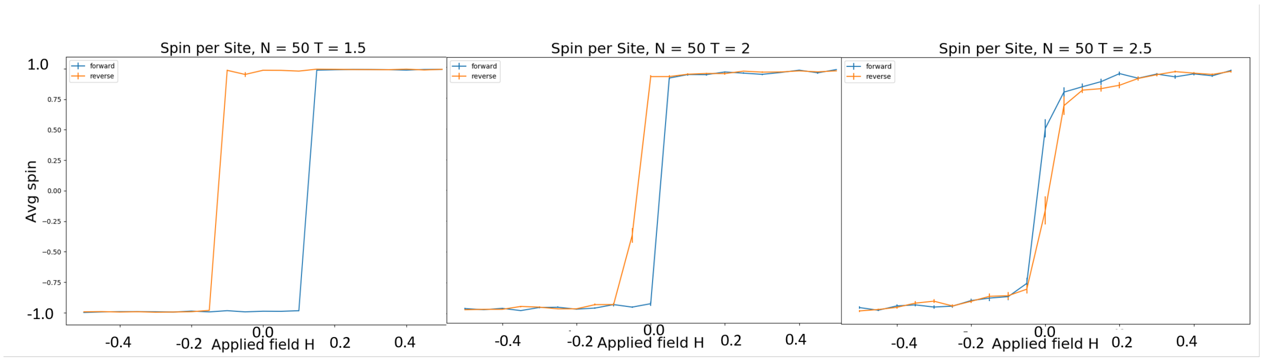


Figure 6: Observations of hysteresis loops produced by incrementally running a random, N = 50 lattice from zero field to the maximum positive field, then recording the result of running back to the max negative before returning to the maximum positive field again. Key observations are the reduction in loop area with increase in temperature.

3.3.2 Loss of Phase Transition

In line with section 1.1, an external field causes the phase transition to become continuous. This effect can be observed in terms of the magnetisation in figure 7.

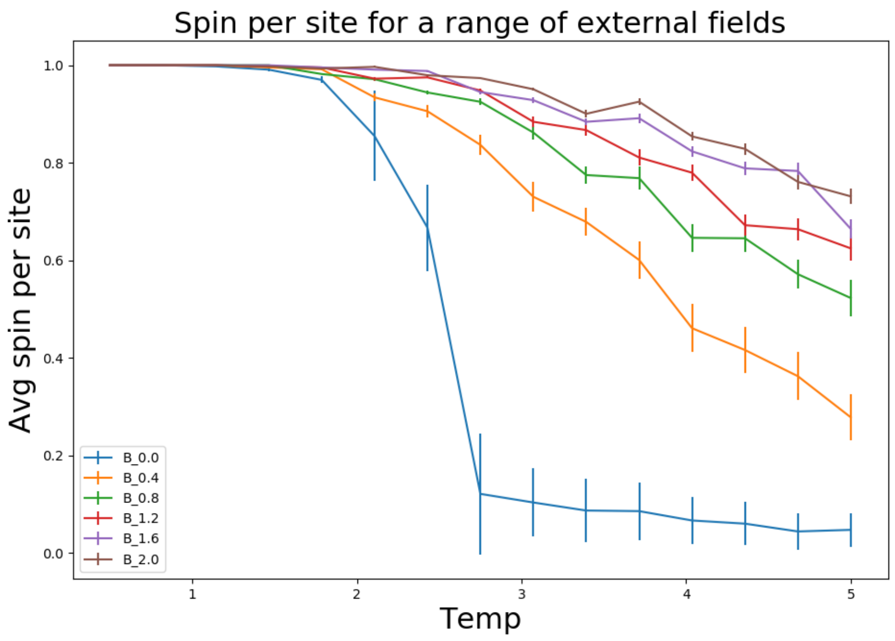
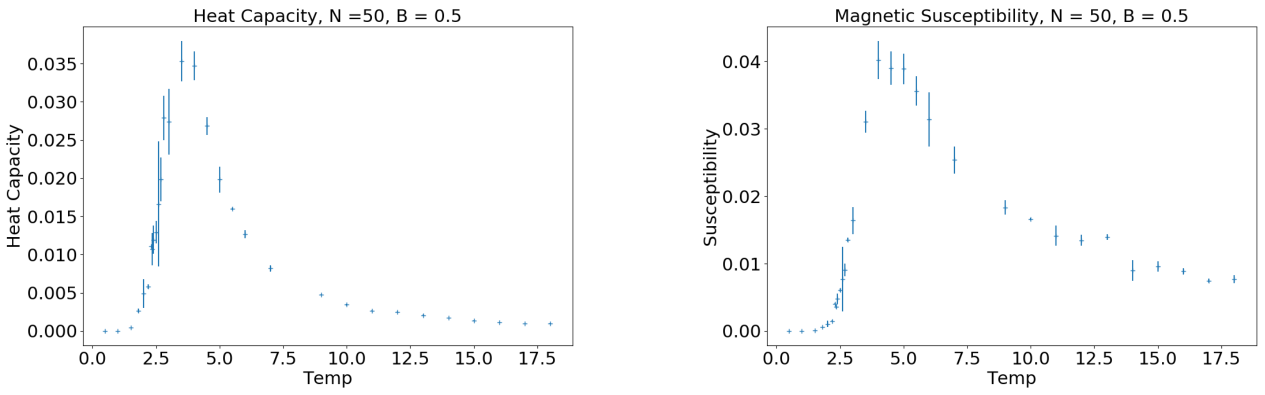


Figure 7: Average spin per site while varying temperature for a range of external field values at $N = 30$. The sharp transition behaviour is lost as field is increased and symmetry is broken.

This effect also causes the loss of divergence of the heat capacity and susceptibility in figure 8. In the presence of a field, the system smoothly undergoes a transition from a low-temperature regime where capacity rapidly grows with temperature to a regime of decaying capacity, at high temperature. These results are discussed further in section 4.2.



(a) Specific heat capacity

(b) Magnetic susceptibility

Figure 8: The behaviour of the system's heat capacity and susceptibility under an external magnetic field for $N = 50$. Observe that the diverging behaviour has been lost, along with the critical phase transition. A smooth transition between the low and high temperature regimes is observed, instead, at a temperature that appears to scale with the field applied.

4 Analysis of results and discussion

4.1 Finite scaling analysis - verification of Onsanger's critical temperature

A key observation from section 3.2 is the dependence of the critical temperature T_C on the lattice size N. A functional form is assumed for this relationship:

$$T_C(N) = T_C(\infty) + aN^{-1/\nu}. \tag{4}$$

As each of the offset and polynomial degree and coefficient for N are unknown in this equation, a regression analysis[2] was undertaken for various ν, to find the best fit of the data in table 3 to a straight line. The result of this analysis can be seen in figure 9.

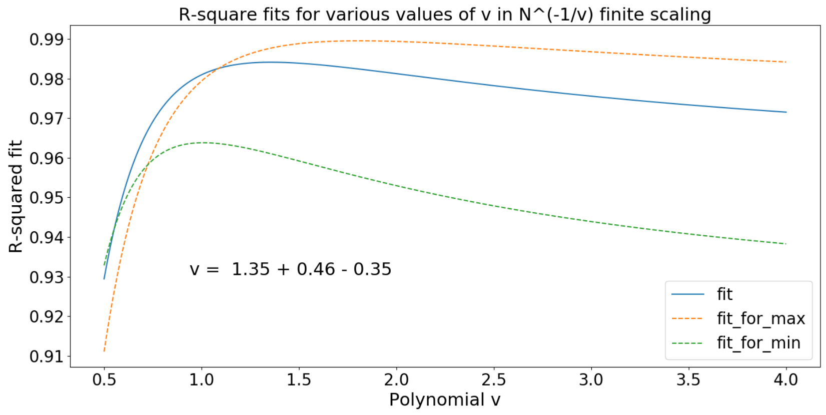


Figure 9: R^2 regression fits to a straight line for varying ν. Results are plotted for the raw T_Cs, along with lines for the standard deviation added to/subtracted from the raw values of T_Cs (max/min). This gives acceptable upper and lower bounds on the value of ν.

Obtaining a value of $\nu = 1.36 + 0.45 - 0.35$, a second regression analysis was performed on the now linear equation 4. By taking the maximum and minimum of each of the variables from minimum and maximum ν respectively, the following results were yielded:

$\nu =$	1.36	$+0.45$	-0.35	
$a =$	2.29	$+1.76$	-0.69	J/k_B
$T_C(\infty) =$	2.23	$+0.04$	-0.05	J/k_B

Table 4: Results of the unknowns in equation 4 from finite scaling analysis.

Quoting Onsanger's analytical result [2] for the critical temperature for $N \to \infty$:

$$T_C(\infty) = \frac{2J}{ln(1+\sqrt{2})k_B} = 2.27(3.d.p)J/k_B \tag{5}$$

The finite scaling result of this computational analysis of the heat capacities is in agreement with the analytical result; thus, the second order phase transition theory is verified by computational methods. It should, however, be noted that the error for the coefficient a is unsatisfactory, due to large uncertainties in peak resolution. Nonetheless, the quantity of interest, $T(\infty)$, is known to sufficient accuracy.

[2]Using NumPy's polyfit for regression data.

4.2 Heat capacity of the spin system with applied field

In figure 10, the temperature for the maximum heat capacity appears to scale with the magnitude of the external field applied. In other words, larger external fields require higher temperatures for the spin-induced heat capacity to decay away.

This information infers that the lattice can only store energy in the spin system while interactions due to alignment with the field (also including self interaction effects at low temperature) are energetically significant compared with thermal energy. The high ($C \to 0$) and low (C rapidly growing) temperature regimes are valid for both magnetic moment and spin flip energy change $|m_0 B|, |\Delta E| << k_B T$ and $|m_0 B|, |\Delta E| >> k_B T$ respectively, much like the hysteresis effect (section 3.3.1). A continuous transition is observed between these regimes..

One further effect observed is the longer retention of energy by larger lattices (i.e. heat capacity decaying only at higher temperatures). It seems plausible that once again, finite scaling effects are acting. These effects appear to reduce the heat capacity of the system when it is more quickly traversed (e.g. smaller in size), which makes physical sense, as a magnetisation change (or loss) can be propagated across the whole lattice more quickly for a given temperature.

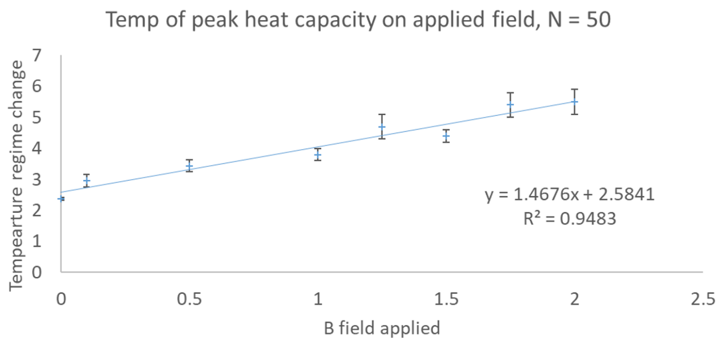


Figure 10: Scaling of capacity peak temperature with field for a lattice of size N = 50.

4.3 Errors

Peak resolution

As is clear in figures such as 5, the largest source of error is the poor behaviour of the finite lattice around the critical point. This made accurate resolution of a peak to high precision difficult, thus limiting the number of data points that could be taken (in conjunction with the computational upper limit on lattice size). An attempt was made to find a peak by simulating as many times as necessary until a peak could be resolved by linear bisection, but due to this erratic behaviour, no clear bisections could be made.

Statistical error

Section 3.2 handled Monte Carlo statistical errors by simulating 3-5 independent lattices per data point. For non critical temperatures, this was found to be satisfactory. Errors in section 4.1's regression analysis were dominated by the large uncertainties in $T_C(N)$ and resulted in very different values of ν, so theoretical errors in these calculations were ignored.

4.4 Improvements for future study

Higher computational power

With more computational power and parallelisation of Python scripts[3], finite-scaling analyses could be done over larger N ranges. With this improvement, more sophisticated simulated annealing analyses could be made to resolve peaks.

Implementing a different algorithm

Improved algorithms for resolving the model around the critical point exist. These rely on flipping clusters of spins, allowing faster non-local moves on energy. Some possibilities are the SwendsenWang algorithm [6] or

[3]Which otherwise execute on a single core.

improved[4] Wolff algorithm [7]. A futher generalisation exists in the Niedermayer's algorithm [8]. All three would improve poor resolution and critical slowing-down [9] around the critical temperatures.

Use of other languages

Python was originally chosen due to its extensive libraries, making computation simpler and more predictable. However, it was noted at various points in the experiment that certain speed-ups could not be accessed with this interpreted language, notably: fixed-length loop compilation (e.g. for four fixed neighbours) and the inevitable slow-down of an interpreted language. Future studies should consider coding in a compiled language, such as C++, with these considerations in mind.

[4]Due to the larger probability of flipping larger clusters.

5 Conclusion

In this quantitative Monte Carlo study of the 2D Ising model, we were able to verify Onsanger's result for the critical temperature for phase transitions for an infinite lattice with no external field, $T_C(\infty) = 2.23 + 0.04 - 0.05 J/k_B$. In pursuit of this value, various predictions of universality of critical components were also verified (for the Ising model).

Furthermore, a study of the lattice's behaviour with an external field applied verified the prediction that the critical phase transition would disappear in this regime. Observations of hysteresis loops indicated that the symmetry of the problem had been broken. Two regimes for the heat capacities and magnetic susceptibilities were observed at high and low temperature, with $C(T \to \infty) \to 0$, and C scaling rapidly with temperature, for low temperature.

The main limitation in the quantitative aspects of this study was the behaviour of a finite lattice around a critical temperature. Further, more accurate computational verifications of Onsanger's theory should employ more powerful computers to reduce statistical errors, or even different algorithms to deal with more nuanced issues discussed in this report.

References

[1] Ernst Ising. Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik*, 31(1):253–258, 1925. `https://doi.org/10.1007/BF02980577`.

[2] Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Phys. Rev.*, 65:117–149, 1944. `https://link.aps.org/doi/10.1103/PhysRev.65.117`.

[3] J. Mohammed and S. Mahapatra. A comparative study of $2d$ Ising model at different boundary conditions using Cellular Automata. *ArXiv e-prints*, 2016. `https://arxiv.org/abs/1601.00518v1`.

[4] H. Nyquist. Thermal Agitation of Electric Charge in Conductors. *Phys. Rev.*, 32:110–113, 1928. `https://doi.org/10.1103/PhysRev.32.110`.

[5] Herbert B. Callen and Theodore A. Welton. Irreversibility and generalized noise. *Phys. Rev.*, 83:34–40, 1951. `https://doi.org/10.1103/PhysRev.83.34`.

[6] Robert H. Swendsen and Jian-Sheng Wang. Nonuniversal critical dynamics in monte carlo simulations. *Phys. Rev. Lett.*, 58:86–88, 1987. `https://link.aps.org/doi/10.1103/PhysRevLett.58.86`.

[7] Ulli Wolff. Collective monte carlo updating for spin systems. *Phys. Rev. Lett.*, 62:361–364, 1989. `https://link.aps.org/doi/10.1103/PhysRevLett.62.361`.

[8] D. Girardi, T. J. P. Penna, and N. S. Branco. Dynamical behavior of the Niedermayer algorithm applied to Potts models. *Physica A Statistical Mechanics and its Applications*, 391:3849–3857, 2012. `https://arxiv.org/abs/1204.4353`.

[9] M. P. Nightingale and H. W. J. Blöte. Dynamic Exponent of the Two-Dimensional Ising Model and Monte Carlo Computation of the Subdominant Eigenvalue of the Stochastic Matrix. *Physical Review Letters*, 76:4548–4551, 1996. `https://arxiv.org/abs/cond-mat/9601059`.

A Appendices

A.1 Metastable states

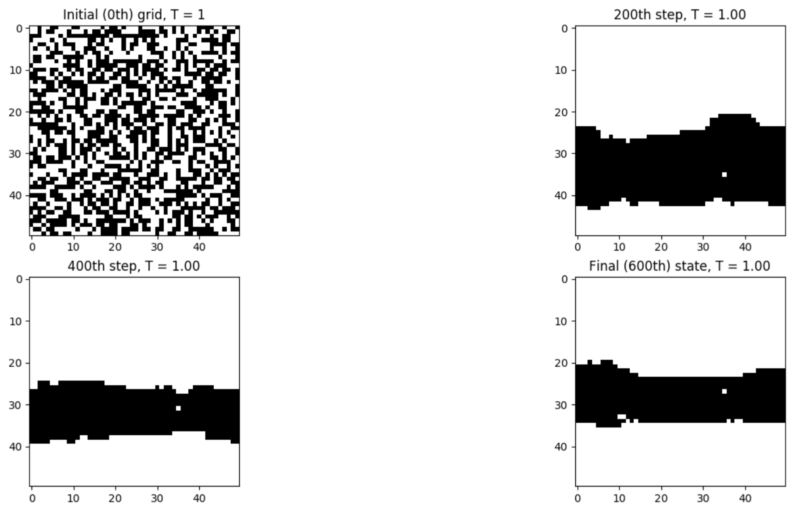


Figure 11: A phenomenon that arose when trying to reach the equilibrium state from an an initially random lattice. The system took a very long time to break this metastability and had the potential to cause some issues.

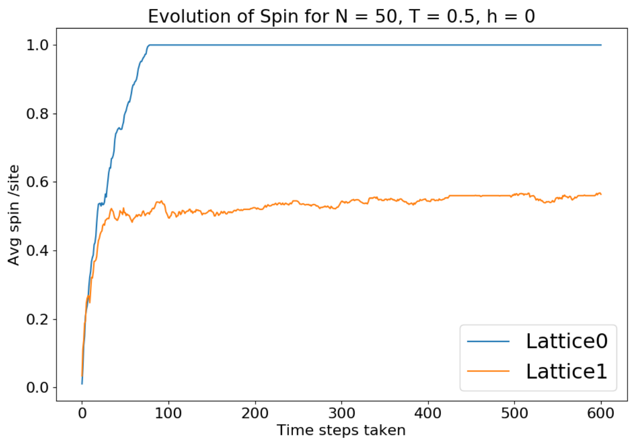


Figure 12: The hindered progression to equilibrium of a system in a metastable state.

A.2 Metropolis Algorithm

Here, we will outline the rules observed by the Metropolis algorithm applied to the Ising model for this study. Below are the steps for a single sweep of the lattice:

1. Pick a spin site at random.
2. Observe (forward-calculated) ΔE- the change in energy of the lattice if this spin were to flip.
3. If $\Delta E < 0$, flip the spin (spontaneously favourable).
4. Otherwise, flip the spin with probability $e^{-\frac{\Delta E}{k_B T}}$ - see A.4 - by comparing with randomly generated number.
5. If the spin was flipped, recalculate ΔE for this site and its neighbours.
6. Repeat until every spin site has been selected only once.

A.3 Typical Lattice Progression

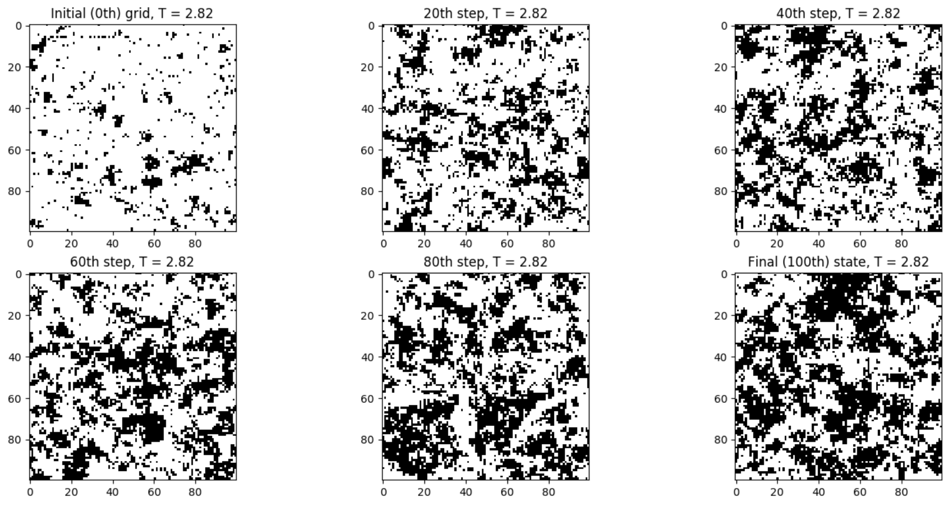


Figure 13: A typical progression of a lattice, close to the critical temperature (phase transition from aligned to random). Note the initial state is the final state of the previous temperature increment, and the final state is used for the following increment.

A.4 Detailed Balance Condition

Detailed balance states, for a system at equilibrium, the forward rate of transition must be equal to the reverse. Thus:

$$\frac{P(i,f)}{P(f,i)} = \frac{\frac{1}{N^2}A(i,f)}{\frac{1}{N^2}A(f,i)} = \frac{A(i,f)}{A(f,i)} = \frac{P_\beta(f)}{P_\beta(i)} = \frac{\frac{1}{Z}e^{-\frac{E_f}{k_B T}}}{\frac{1}{Z}e^{-\frac{E_i}{k_B T}}} = e^{-\frac{\triangle E}{k_B T}} \tag{6}$$

- P(i,f) - probability of transitioning from state i(nitial) to a new state f(inal).
- $\frac{1}{N^2}$ - is the selection probability of any site. This is the condition that requires selection to be randomised (in order to cancel).
- Acceptance rate between states A
- $P_\beta(a)$ - Boltzmann probability of finding the site in state a.

Thus it is seen directly that the acceptance probability:

$$\frac{A(i,f)}{A(f,i)} = e^{-\frac{\triangle E}{k_B T}} \tag{7}$$

due to the detailed balance condition.

B Code Listing

Here the Python main script (e.g. the script that completed the bulk of the calculation) will be displayed. Experiments imported this script for use in various ways. For brevity, these will be omitted, but are available on request.

```python
import sys, random, time, copy
import numpy as np
import scipy.constants as const
import matplotlib as mpl
from matplotlib import pyplot as plt

"""Main script for the ising model simulation
        Run this script with parameters (at bottom of file) to progress a single
            lattice
        Selected number of times, temperature, field etc
        Prints the results to a heatmap figure to observe, amongst other
            functionality"""

mu = 1 #Mag permeability of free space
mu_e = 1 #const.physical_constants["electron mag. mom."][0] #magnetic moment of an
    electron
h_bar = 1 #const.hbar
J_e = 1 #spin exchange energy
k_b = 1 #const.k # boltzman constant

class lattice_site :
        """A class for storing a spin site with its interactions with its
            neighbours"""

        def __init__(self, row, col, spin = 0, nearest_neighbours=[]) :
                """Initialises a lattice site with random spin (unless specified
                    as +/-1)"""

                self.row, self.col = row, col

                self.nearest_neighbours = nearest_neighbours

                if spin == 1 or spin == -1  :

                        self.spin = spin

                else :
                        if spin != 0 :
                                print("Warning: spin != 0, read " +str(spin), end=
                                    "\r")

                        random_number = np.random.random_sample()

                        if random_number < 0.5  :
                                self.spin = -1
                        else :
                                self.spin = +1

                self.alt_energy = -1

        def flip_spin(self, temp, log_rand_num_kbt, field, exch_energy, mag_moment
            ) :
                """If favourable, spin is flipped and energy/spin changes returned
                Calculates alternative energy and progresses if < 0 or boltz prob
                Implementation of metropolis algorithm"""
                flipped, dE, dMag = False, 0, 0

                #If not favourable, flip with boltz probability. Else flip (if
                    favourable)
                if self.alt_energy <= 0 \
                        or (self.alt_energy > 0 and log_rand_num_kbt <  -1. * self
                            .alt_energy ):

                        dMag = -2 * self.spin
                        self.spin *= -1
                        dE = self.alt_energy
                        flipped = True

                return flipped, dE, dMag

        def calculate_energy(self, field, exch_energy, mag_moment) :
                """Calculate and return the energy contribution to the lattice by
                    this spin site
                Dependent on field and nearest neighbour spin interactions."""

                #find the spin interactions
                spin_interaction = 0

                for neighbour in self.nearest_neighbours :

                        spin_interaction += -1. * exch_energy * neighbour.spin *
                            self.spin * h_bar ** 2

                #Find the field contribution
                field_contribution = -1. * self.spin * h_bar * mag_moment * field

                #print("field cont", field_contribution)
                #print("field", field)

                return spin_interaction + field_contribution

        def calculate_alt_energy(self, field, exch_energy, mag_moment):

                self.alt_energy = \
                 -2 * self.calculate_energy(field= field, exch_energy =
                    exch_energy, mag_moment = mag_moment)

        def get_spin(self) :
                """Return the spin of the site"""

                return self.spin

        def get_neighbour_locs(self, N) :

                prev_row = (self.row - 1) % N
                next_row = (self.row + 1) % N

                prev_col = (self.col - 1) % N
                next_col = (self.col + 1) % N

                return prev_row, next_row, prev_col, next_col

        def set_neighbours(self, neighbour_list) :
                """Set the nearest neighbours"""

                self.nearest_neighbours = neighbour_list
```

```
class lattice :
        """Stores an NxN lattice_array of 'lattice_site's and has a number of
            useful functons pertaining to a whole lattice"""

        def __init__(self, N, T, field = 0, exch_energy = J_e, mag_moment = mu_e,
            typ = "r") :
                """Make an NxN lattice of lattice_site s
                type corresponds to initial spin assignment (r for random)
                returns an NxN dictionary of lattice sites keyed by [x][y] integer
                     location"""

                #Lattice parameters
                self.size = N
                self.T = T
                self.field = field
                self.exch_energy = exch_energy
                self.mag_moment = mag_moment

                #Set type of array/spin
                if typ == "r" :
                        spin = 0
                elif typ == "a+" :
                        spin = 1
                elif typ == "a-" :
                        spin = -1
                else :
                        print("Lattice␣type", typ, "not␣understood␣in␣
                            initialisation.")
                        print("Try␣r,␣a+,␣a-.")
                        sys.exit()

                #Make and set the lattice array
                self.lattice_array = np.empty((N,N), dtype=lattice_site)

                for row in range(N) :
                        for col in range(N) :
                                self.lattice_array[row, col] = lattice_site(row,
                                    col, spin=spin)

                #Set the neighbours of the lattice sites
                self.set_all_neighbours()

                #Save the current alternative energies (e.g. if spin flipped)
                for row in range(N) :
                        for col in range(N) :
                                self.lattice_array[row, col].calculate_alt_energy(
                                    field= field, exch_energy = exch_energy,
                                    mag_moment = mag_moment)

                #Calculate set of random numbers once only for each lattice
                self.rand_array = np.log(np.random.rand(N**2))* k_b * self.T

                #Find initial properties
                self.net_energy = self.get_net_energy()
                self.net_spin = self.get_net_spin()

        def new_run(self, T, field = 0, exch_energy = J_e, mag_moment = mu_e) :
                """Uses the same lattice but updates the temp, field and random
                    numbers"""

                self.T = T
                self.field = field
            self.exch_energy = exch_energy
            self.mag_moment = mag_moment

            #Calculate new set of random numbers for this run
            self.rand_array = np.log(np.random.rand(self.size**2))* k_b * self
                .T

    def permute_rand_array(self) :
            #Iterate through the random list , permute each by 1

            np.random.shuffle(self.rand_array)

    def set_all_neighbours(self) :
            """Sets all the lattice_array's neighbour lists to the appropriate
                neighbours
            for every lattice site in the lattice"""

            for row in range(self.size) :
                    for col in range(self.size) :

                            #Get locations
                            prev_row, next_row, prev_col, next_col = self.
                                lattice_array[row, col].get_neighbour_locs(self
                                .size)

                            #Find neighbours
                            neighbours = [self.lattice_array[prev_row, col],
                                self.lattice_array[next_row, col], self.
                                lattice_array[row, prev_col], self.
                                lattice_array[row, next_col]]

                            #set neighbours
                            self.lattice_array[row, col].set_neighbours(
                                neighbours)

    def progress_n(self, n, graphs = 0) :
            """Progresses the process by n steps- graphs initial, every '
                display'th and final
            returns list of lines of steps for later graphing
            step, energy, spin"""

            lns = []

            #List of locations to progress
            locs = self.get_all_locations()

            plot_row, plot_col, mid_graphs, axarr = self.initialise_figure(n,
                graphs)

            #Print the initial state
            print("initial net lattice energy, spin:", self.net_energy, self.
                net_spin)

            #Iterate + progress for n steps
            for i in range(n) :

                    #Add plot to graphs if in list
                    if i in mid_graphs :
                            self.add_to_plots(axarr[plot_row, plot_col])
                            axarr[plot_row, plot_col].set_title(str(i) + "th
                                step, T = " + ("%.2f" % self.T))
                            plot_col += 1
                    if plot_col >= graphs/2 :
                            plot_row += 1
                            plot_col = 0

            #Add to output lines
            lns.append( str(i) + ", " +  str(self.net_energy) + ", " +
                 str(self.net_spin/(self.size **2)))

            #Shuffle locations and progress
            random.shuffle(locs)
            flip_perc = self.progress(locs)

            #Track progress
            print(i + 1, "/", n, "progress. Flipped", flip_perc, "% of
                 sites", end="\r" )


        #Append the final state
        lns.append( str(i+1) + ", " +  str(self.net_energy) + ", " + str(
            self.net_spin/(self.size ** 2)))

        #Plot the final state
        if graphs != 0 :
                heatmap = self.add_to_plots(axarr[plot_row, plot_col])
                axarr[plot_row, plot_col].set_title("Final (" + str(i+1) +
                     "th) state, T = " + ("%.2f" % self.T))
                #plt.colorbar(heatmap, ticks=[-1,1])

        #Print the final state
        print()
        print("Final (" + str(i+1) + "th) net lattice energy, spin:", self
            .net_energy, self.net_spin)

        return lns

def progress(self, locs) :
        """Progresses the lattice by stepping through a random row, col
        Then use lattice_site check spin flip method. Updates the net
            energy and spin also"""

        #print("locs", locs)
        flip_count, tot = 0, 0

        #Update each grid cell in a random order
        for loc in locs :

                row, col = loc[0], loc[1]

                flipped, dE, dM = \
                        self.lattice_array[row, col].flip_spin(\
                         self.T, self.rand_array[row*self.size + col],
                            field=self.field, exch_energy = self.
                            exch_energy, mag_moment = self.mag_moment)

                tot += 1

                #Detect a spin flip
                if flipped :
                        flip_count += 1

                        #Update the alt_energy array for cell and
                            neighbours
                    prev_row, next_row, prev_col, next_col = \
                     self.lattice_array[row,col].get_neighbour_locs(
                        self.size)

                    self.lattice_array[row, col].calculate_alt_energy(
                        field= self.field, exch_energy = self.
                        exch_energy, mag_moment = self.mag_moment)
                    self.lattice_array[prev_row, col].
                        calculate_alt_energy(field= self.field,
                        exch_energy = self.exch_energy, mag_moment =
                        self.mag_moment)
                    self.lattice_array[next_row, col].
                        calculate_alt_energy(field= self.field,
                        exch_energy = self.exch_energy, mag_moment =
                        self.mag_moment)
                    self.lattice_array[row, prev_col].
                        calculate_alt_energy(field= self.field,
                        exch_energy = self.exch_energy, mag_moment =
                        self.mag_moment)
                    self.lattice_array[row, next_col].
                        calculate_alt_energy(field= self.field,
                        exch_energy = self.exch_energy, mag_moment =
                        self.mag_moment)

                self.net_energy += dE
                self.net_spin += dM

        #print(self.rand_array)
        self.permute_rand_array()

        flip_perc = int(round(100 * flip_count/tot))

        return flip_perc

    def get_all_locations(self):
        """Return a list of random locations in the grid"""

        locs = []

        #Shuffle the order of the rows and cols
        rows, cols = list(range(self.size)), list(range(self.size))

        for row in rows :
            for col in cols :
                locs.append((row, col))

        return locs

    def initialise_figure(self, n, graphs) :
        """Initialises the plot space for plotting of the outputs
        Defaults to plotting 1x2 or 2x(g/2) graphs"""

        if graphs == 0 :
            return 0, 0, [], []

        #Make even
        if graphs % 2 != 0 :
            graphs -= 1

        #Make the figure
        if graphs == 2 :
            fig, axarr = plt.subplots(1, 2)
            else :
                    fig , axarr = plt.subplots(2, int(graphs/2))

            #Start after initial , set other plot positions
            plot_row , plot_col = 0, 1
            mid_graphs = (np.round(np.linspace(0, n, num = graphs))[1:]).
                astype(int)

            #Plot initial
            self.add_to_plots(axarr[0,0])
            axarr[0,0].set_title("Initial␣(0th)␣grid,␣T␣=␣" + ("%.2f" % self.T
                ))

            return plot_row , plot_col , mid_graphs , axarr

    def add_to_plots(self , ax) :
            """"Plots the matrix onto an axis of a subplot , passed as an
                argument"""

            #get a matrix of +/- 1 values
            v_spin = np.vectorize(lattice_site.get_spin)
            row_col_matrix = v_spin(self.lattice_array)

            #ax.set_xlabel("Col positions", labelpad=-3.5)
            #ax.set_ylabel("Row positions")

            #Heatmap customisations
            cmap= mpl.colors.ListedColormap(['k', 'w'])
            bounds = [-1, 0, 1]
            norm= mpl.colors.BoundaryNorm(bounds, cmap.N)
            heatmap = ax.imshow(row_col_matrix , cmap=cmap, interpolation='none
                ', norm=norm)

            return heatmap

    def get_net_energy(self) :
            """Return the net magnetic energy in the lattice"""

            #Define vectorisation
            v_calc = np.vectorize(lattice_site.calculate_energy)

            #Grid of energy contributions for each site
            energies = v_calc(self.lattice_array , self.field , self.exch_energy
                , self.mag_moment)

            return energies.sum()

    def get_net_spin(self) :
            """Return the spin in the lattice ,avg per site"""

            v_spin = np.vectorize(lattice_site.get_spin)
            spins = v_spin(self.lattice_array)

            return spins.sum()

def print_lines_to_file(filename , lines) :
        """Prints the lines passed to a file"""

        print("Writing␣lines␣to␣file", filename , "in␣cwd.")
        with open(filename , 'w') as f :

                for row in lines :
                    f.write(row + "\n")

        print("Success")

def read_in_from_file(filename) :

        print("Reading from", filename, "in cwd.")

        lines = []

        with open(filename, 'r') as f :

                for line in f :
                        lines.append(line.strip())

        print("Success.")

        return lines

def plot_one(rows, energy = True) :
        """Plots energy (true) or spin from rows output on time steps"""

        plt.figure()

        if energy :
                tit = "energy"
        else :
                tit = "spin"

        plt.title("Evolution of net lattice " + tit + " with time.")
        plt.xlabel("Time steps taken")

        for i in range(len(rows)):

                splt = rows[i].split(", ")

                for el in range(len(splt)) :
                        splt[el] = float(splt[el])

                rows[i] = splt

        rows = np.array(rows)

        steps = rows[:,0]
        energies = rows[:,1]
        log_energies = np.log(abs(energies))
        spins = rows[:,2]
        log_spins = np.log(abs(spins))

        if energy :
                plt.ylabel("Lattice Energy / Joules/(exchange_energy*h_bar^2)")
                plt.plot(steps, energies, 'rx')
                #plt.figure()
                #plt.plot(steps[1:], log_energies[1:])
        else :
                plt.ylabel("Net integer spin")
                plt.plot(steps, spins, 'b*')

def plot_both(rows) :
        """Takes energy, spin strings and plots on steps on a two-axis figure"""

        ax1 = (plt.figure()).add_subplot(111)
        ax2 = ax1.twinx()

        ax1.set_title("Evolution of net lattice energy and spin with time.")
        ax1.set_xlabel("Time steps taken")

        for i in range(len(rows)):

                splt = rows[i].split(", ")

                for el in range(len(splt)) :
                        splt[el] = float(splt[el])

                rows[i] = splt

        rows = np.array(rows)

        steps = rows[:,0]
        energies = rows[:,1]
        spins = rows[:,2]

        ax1.set_ylabel("Lattice Energy / Joules/(exchange_energy*h_bar^2)")
        ax1.tick_params('y', colors='b')
        ax1.plot(steps, energies, 'rx')


        ax2.set_ylabel("Net integer spin")
        ax2.plot(steps, spins, 'b*')

if __name__ == "__main__" :

        ##################################################
        ############# runtime parameters ###############

        #size of lattice NxN
        N = 50

        #type of lattice (r andom, a+ a- ligned), Temp in J_e/k_b (set = 1)
        typ = 'r'
        temp = 2.5

        #field
        h = 0

        #steps - whole lattice progressions
        n = 50

        #graphing

        #number of graphs to display
        graphs = 0
        ##################################################

        #Init lattice
        print("Initialising...")
        init_start = time.clock()

        lattice = lattice(N, temp, field = h, exch_energy = J_e, mag_moment = mu_e
            , typ=typ)

        init_end = time.clock()

        print()
```

```
    print(init_end-init_start, "seconds to initialise")
    print()

    #Progress the lattice n times
    lines = lattice.progress_n(n, graphs=graphs)

    end = time.clock()

    print()
    print(end - init_end, "seconds to progress", n, "times.", N,"x",N, "grid")
    print((end-init_end)/n,"seconds per step.")
    print()

    plt.show()
```