CO28: Simulation of the Ising model for ferromagnetism using the Metropolis algorithm

Richard K. Aw (corp2841)

Corpus Christi College, University of Oxford

March 6, 2020

Abstract

The two-dimensional square lattice Ising model for ferromagnetism was studied via a computational approach involving the implementation of the Metropolis algorithm. It was found that in most cases the 'magnetisation' of the square lattice typically underwent a sharp change before converging to a stable value after a certain number of 'sweeps', in accordance with the phenomenon of a phase transition. Some numerical results (e.g., the critical value at which the phase transition occurs) were also obtained, and these results were found to be in good agreement with theoretical predictions.

Introduction

Ferromagnetism is a phenomenon where certain materials exhibit spontaneous magnetisation — a net magnetic moment in the absence of an external magnetic field. It is really a macroscopic phenomenon that emerges from the microscopic behaviour of and interactions among individual electrons, whose magnetic dipole moments become aligned in a parallel manner to give rise to the magnetisation. Moreover, the magnetic dipole moment of an electron arises from a more fundamental, intrinsic property of the electron — spin angular momentum, or 'spin'. (When measured, the spin of an electron can take only one of two possible values: 'up' or 'down'.) Thus, ferromagnetism can really be conceived as the parallel alignment of spins, which occurs fundamentally because the spin of any particular electron tends to align (in a parallel manner) with the spins of its neighbouring electrons and with an external magnetic field (if any). This leads to the idea of simulating ferromagnetism using the Ising model.

The Ising model is a mathematical model of ferromagnetism which, when applied to a two-dimensional square lattice of spins, is characterised by the following equation [1]:

$$E = -J \sum_{i,j} S_i S_j - B \sum_i S_i \tag{1}$$

The terms in equation (1) may be understood as follows: E denotes the energy of the lattice, J is a parameter governing the strength of *exchange interactions* between pairs of neighbouring spins and whose polarity determines whether the lattice is ferromagnetic (if $J > 0$) or antiferromagnetic (if $J < 0$), B is a parameter governing the strength of an external magnetic field applied to the lattice, and S_k denotes the k^{th} spin (either $+1$ for 'up' or -1 for 'down'). There are two caveats, though, that the notation in the equation does not make apparent. First, while it is clear that the second sum on the R.H.S. of equation (1) is over all lattice sites (i.e., it is the sum of all of the spins in the lattice), the first sum is over all pairs of nearest-neighbouring (i.e., adjacent) spins only, not all possible pairs of spins. The second caveat, which builds on the first, is that the pairs are *non-ordered*, so for example, S_2S_3 is the same as S_3S_2 and the pair is counted only once in the sum.

We can understand how equation (1) describes ferromagnetism by considering that any system naturally tends to the lowest energy configuration (*energy minimisation principle*), and two observations about the equation. First, for a ferromagnetic (so $J > 0$) system, its energy E is minimised when as many pairs of adjacent spins $\{S_i, S_j\}$ as possible are aligned (i.e., each of these pairs is either $\{+1, +1\}$ or $\{-1, -1\}$). This is consistent with the idea mentioned earlier that ferromagnetism can fundamentally be thought of as the parallel alignment of spins. Second, E is reduced further when the sum of spins (or, equivalently, the magnetisation of the ferromagnetic system) has the same polarity as B (assuming $B \neq 0$). This is consistent with the fact that when an external magnetic field is applied to a ferromagnetic object, the object adopts a magnetisation that points in the same direction as the magnetic field.

Methodology

A Python script was written to implement the two-dimensional square lattice Ising model. The script basically does the following (in order):

1. Prepares an array of elements with randomly assigned values of ± 1. The array and its elements represent the lattice and the spins of its constituent particles.
2. 'Sweeps' across the entire array (via a 'sweep function') over a certain number of iterations ('sweep cycles'). The sweep function is essentially an implementation of the Metropolis algorithm; it governs the *dynamics* of the array. The values of its internal parameters may be altered to observe corresponding changes in the dynamics of the array.
3. Outputs plots that provide insight into the Physics of ferromagnetism.

Implementing the Metropolis algorithm

How the sweep function works is best illustrated with the following flowchart:

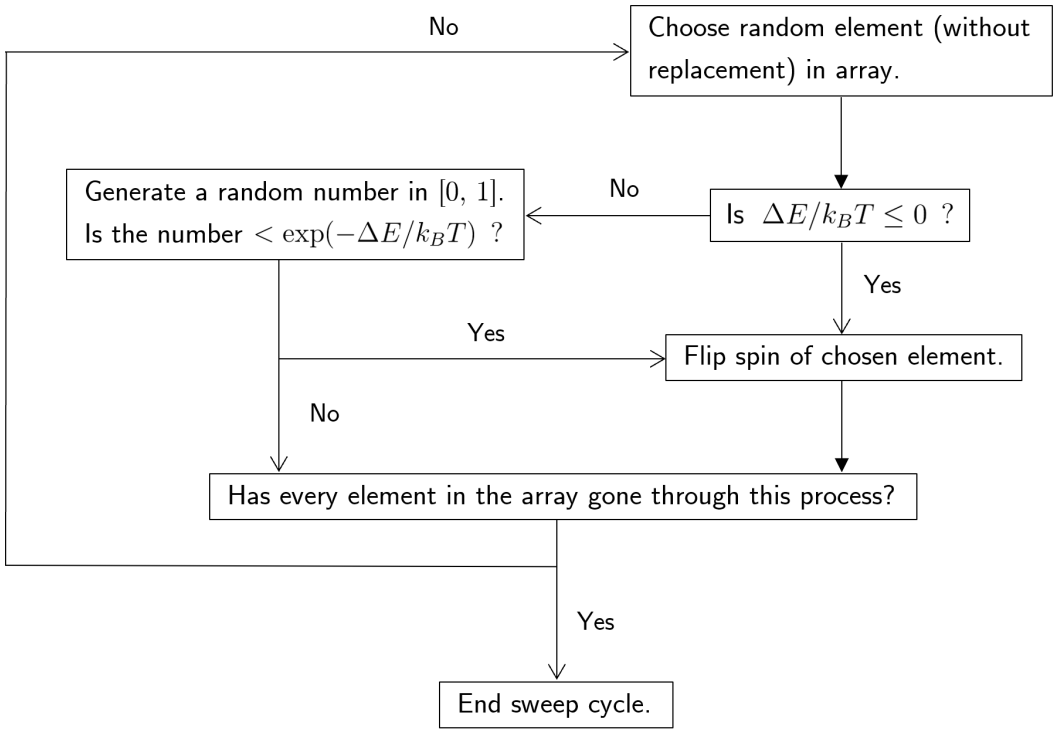

Figure 1: Mechanism of one sweep cycle.

A working explanation of the Metropolis algorithm — what it is, why it is used in this context (i.e., what purpose it serves), and how it serves said purpose — is provided below.

- **Why:** An integral objective of this study is to simulate ferromagnetism, the phenomenon where most of the spins attain parallel alignment. As such, we want to get from the initial state of the array, where the elements are assigned values of $+1$ or -1 randomly, to the desired ferromagnetic state, where most of the elements are either $+1$ or -1. The Metropolis algorithm enables this transition to be made by providing a rule for deciding whether a spin should or should not be flipped. The decision rule is then applied to every element of the array, thus completing a sweep cycle. The sweep cycle is repeated over multiple iterations, thereby allowing the array to make the eventual transition to the desired ferromagnetic state.

- **What:** The Metropolis algorithm is a special case of the Metropolis-Hastings (M-H) algorithm, which in turn belongs to a class of algorithms known as Markov Chain Monte Carlo (MCMC) methods. MCMC methods are used for sampling from probability distributions. They are very useful (from a computational perspective) because drawing samples *directly* from a probability distribution can be computationally expensive, even if its probability density function is known to us [2].

  ▶ In this context, the M-H algorithm would work like this. Suppose the array is in a configuration $\mathbf{S}_t$ at some time step t. We then consider some alternate configuration $\mathbf{S}'$ which the array could transition to, and the proposal density $Q(\mathbf{S}'|\mathbf{S}_t)$ associated with it. Next, we compute the quantity r given below, where $P(\mathbf{S})$ denotes the probability that the array is in the configuration $\mathbf{S}$:

$$r = \frac{P(\mathbf{S}')\, Q(\mathbf{S}_t|\mathbf{S}')}{P(\mathbf{S}_t)\, Q(\mathbf{S}'|\mathbf{S}_t)} \tag{2}$$

  The algorithm then says this (the decision rule). If $r \geq 1$, make the transition. Otherwise, make the transition with probability r. In any case, if the transition is made, set $\mathbf{S}_{t+1} = \mathbf{S}'$ (i.e., the alternate configuration becomes the configuration at the next time step). If the transition is not made, set $\mathbf{S}_{t+1} = \mathbf{S}_t$ (i.e., the current configuration becomes the configuration at the next time step). Put in more succinct terms, the M-H algorithm supplies the following decision rule. Make the transition from $\mathbf{S}_t$ to $\mathbf{S}'$ with a probability given by:

$$P(\mathbf{S}_t \to \mathbf{S}') = \min(1,\ r) \tag{3}$$

  ▶ The M-H algorithm reduces to the Metropolis algorithm when the proposal density is symmetric, that is, $Q(\mathbf{S}_t|\mathbf{S}') = Q(\mathbf{S}'|\mathbf{S}_t)$. Thus, equation (2) can be rewritten as $r = P(\mathbf{S}')/P(\mathbf{S}_t)$. But we also know from statistical mechanics that $P(\mathbf{S})$ follows a Boltzmann distribution: $P(\mathbf{S}) \propto \exp(-E(\mathbf{S})/k_B T)$. Hence, $r = \exp(-\Delta E/k_B T)$, which we substitute into equation (3) to get the transition probability:

$$P(\mathbf{S}_t \to \mathbf{S}') = \min(1,\ e^{-\Delta E/k_B T}) \tag{4}$$

  Equation (4) is the defining equation of the Metropolis algorithm in the context of this study. We now have to figure out how to implement this transition probability in the sweep function.

- **How:** An element is selected from the array randomly and without replacement. To make the spin S of the element flip with the transition probability, we have to calculate ΔE (or $\Delta E/k_BT$ in practice). $\Delta E/k_BT$ may be calculated via the following equation[1]:

$$\frac{\Delta E}{k_BT} = 2 \cdot \frac{J}{k_BT} \cdot S \sum S_{\text{adjacent}} \; + \; 2 \cdot \frac{B}{k_BT} \cdot S \tag{5}$$

It is important to appreciate that equation (5) is consistent with equation (4) and Physics. Consider the spin S. If the sum of its immediately neighbouring spins, $\sum S_{\text{adjacent}}$, has an opposite polarity, then equation (5) tells us that it is highly likely[2] that $\Delta E/k_BT \leq 0$ (i.e., the proposed transition reduces or keeps constant the lattice's energy). This implies that $\exp(-\Delta E/k_BT) \geq 1$ which, according to equation (4), means the transition probability would be 1. That is, S *will* be flipped. This is consistent with Physics since transitioning to a configuration of lower energy is always favourable, according to the energy minimisation principle.

Conversely, if $\sum S_{\text{adjacent}}$ has the same polarity as S, then equation (5) tells us that it is highly likely that $\Delta E/k_BT > 0$ (i.e., the proposed transition increases the lattice's energy). This implies that $\exp(-\Delta E/k_BT) < 1$ which, according to equation (4), means the transition probability would be $\exp(-\Delta E/k_BT)$. That is, S is flipped with probability $\exp(-\Delta E/k_BT) < 1$.★ This is consistent with Physics since transitioning to a configuration of higher energy is not favourable. Furthermore, the greater the value of ΔE, the lower the transition probability $\exp(-\Delta E/k_BT)$ — the greater the 'jump' in energy, the less likely the transition would occur. Another way to think about this is that if a spin is aligned with most of its neighbouring spins, it would *resist* being flipped.

★ This can be implemented by generating a random number N on the interval $[0, 1]$, then making the spin flip if $\exp(-\Delta E/k_BT) > N$.

Imposing periodic boundary conditions

Besides increasing the size of the lattice, another way to boost the accuracy of the simulation's results is to maximise the interactions of the spins at the edges of the lattice. This can be achieved by making said spins interact with those spins at geometrically opposite edges of the lattice. This is done with periodic boundary conditions, which can be imposed on an `m`×`m` lattice by specifying the locations of spins immediately neighbouring a spin at some location `(i, j)` as `((i+1)%m, j)`, `(i, (j+1)%m)`, etc., instead of `(i+1, j)`, `(i, j+1)`, etc., where `%` denotes the modulus operator. A nice way to visualise how the lattice is changed by the periodic boundary conditions would be to consider the (two-dimensional) lattice being folded into a (three-dimensional) torus, with the spins situated on its surface [3].

---

[1] This can be derived from equation (1). A subtle but crucial step to make in the derivation is to consider that when the spin S is flipped (from $+1$ to -1 or vice versa), the change in the lattice's energy is proportional to twice the current spin S.

[2] This is definitely true insofar as the value of B/k_BT is sufficiently small compared to the value of J/k_BT.

Results

A 30×30 lattice of spins ± 1 was created and used for all simulations. To produce preliminary plots[3] of the lattice's magnetisation for various values of J/k_BT and B/k_BT, the sweep function was first ran over 50 cycles to 'wash out' the effects of the initial configuration [4]. It was then ran over 1000 cycles, which ensured that the lattice settled into its equilibrium configuration. Below are some of the preliminary plots, with relevant analyses where applicable.

Preliminary Plots

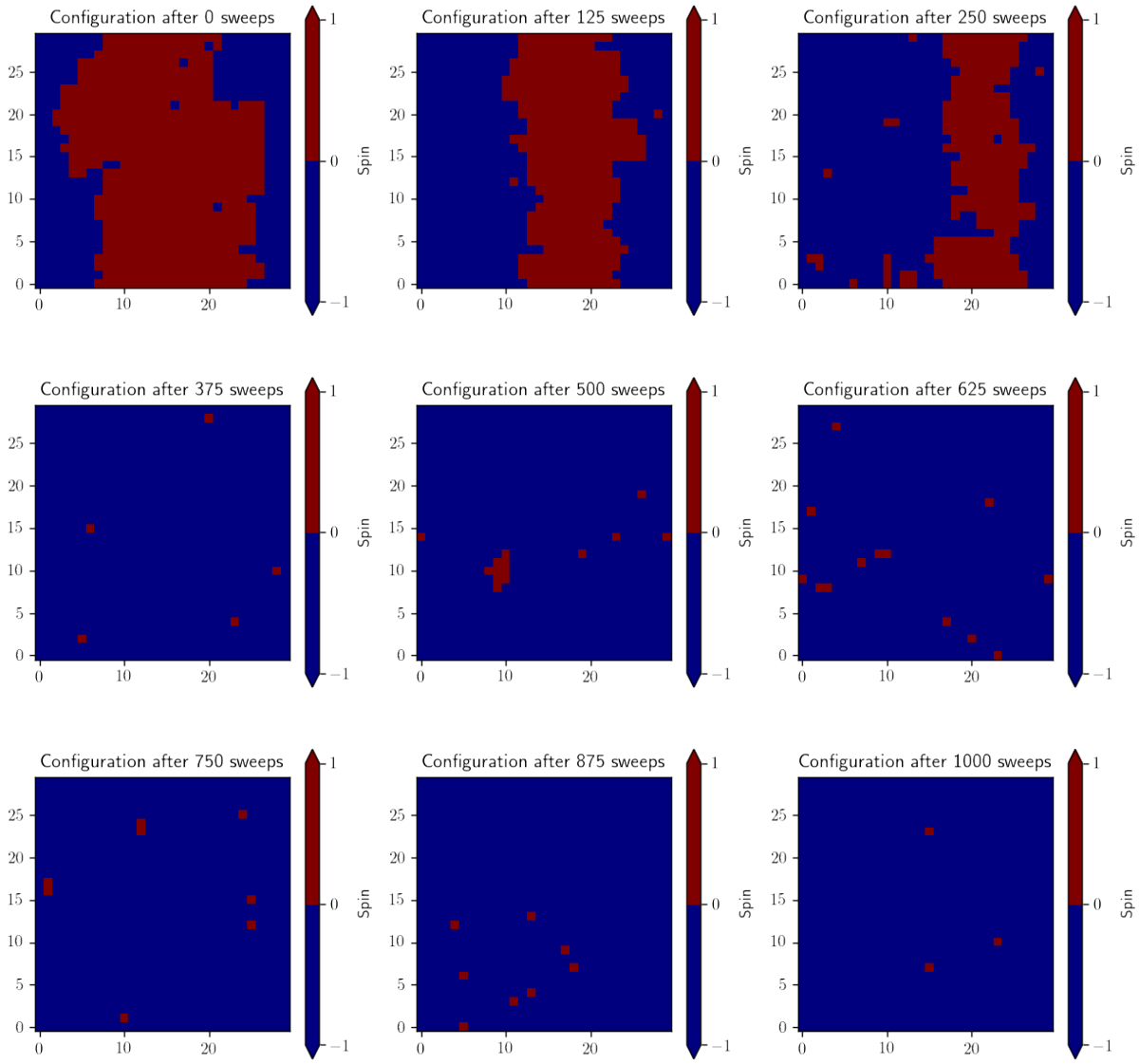

Figure 2a: Colour maps showing the lattice's configuration over 1000 sweep cycles for $J/k_BT = 0.6$ and $B/k_BT = 0.001$, in intervals of 125 sweep cycles.

[3] These are necessary for checking that the lattice is changing in a sensible way.

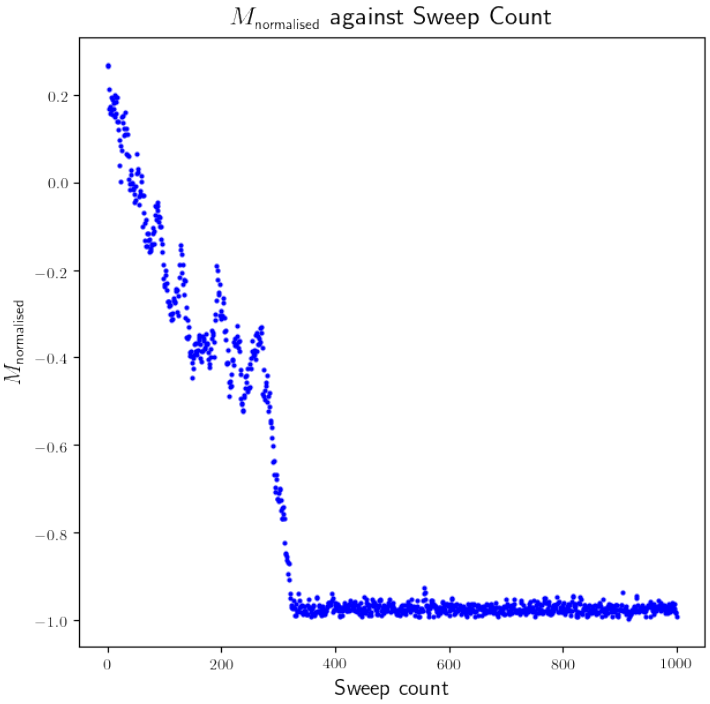

Figure 2b: Scatter plot of the lattice's magnetisation (normalised) against the sweep cycle number for $J/k_BT = 0.6$ and $B/k_BT = 0.001$. It is clear that the magnetisation converges to a narrow range of values after undergoing fluctuating changes in value over the initial sweeps.

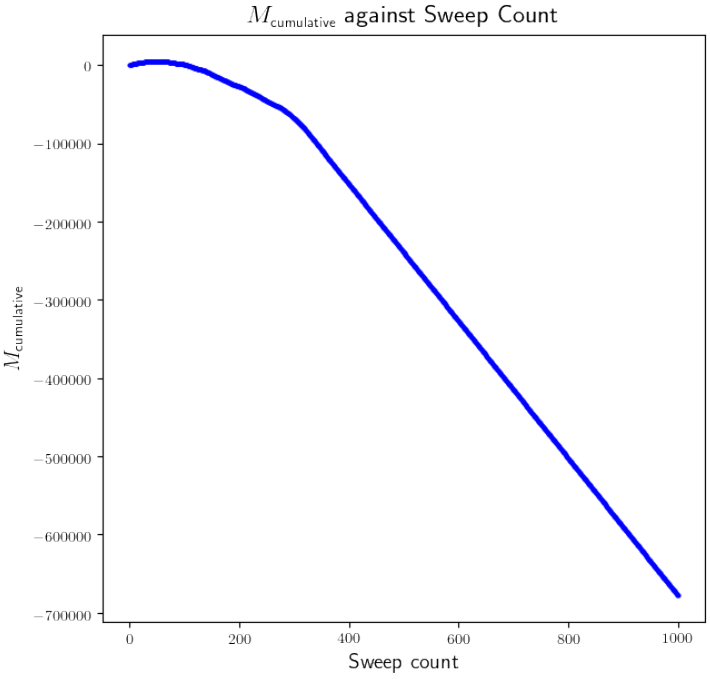

Figure 2c: Scatter plot of the lattice's cumulative magnetisation against the sweep cycle number for $J/k_BT = 0.6$ and $B/k_BT = 0.001$. The cumulative magnetisation changes non-uniformly over the initial sweeps (corresponding to the fluctuating changes in the magnetisation over the initial sweeps), then decreases with a constant gradient (corresponding to the magnetisation's convergence to a narrow range of values).

Similar preliminary plots were produced for various values of J/k_BT and B/k_BT. Most of them displayed the same trend where the magnetisation underwent a non-uniform change in value over the initial sweeps before converging to a stable, narrow range of values. Those that did not display the same trend showed patterns that were within expectations nonetheless. For example, for $J/k_BT = 0$ and $B/k_BT = 0$ the magnetisation alternated between two constant values, one being the negative of the other. This is unsurprising since $J/k_BT = 0$ and $B/k_BT = 0$ would imply that $\Delta E/k_BT = 0$, and given the decision rule implemented, this means that *every* spin in the lattice would be flipped after the passing of *each* sweep cycle. A few other examples are provided overleaf.

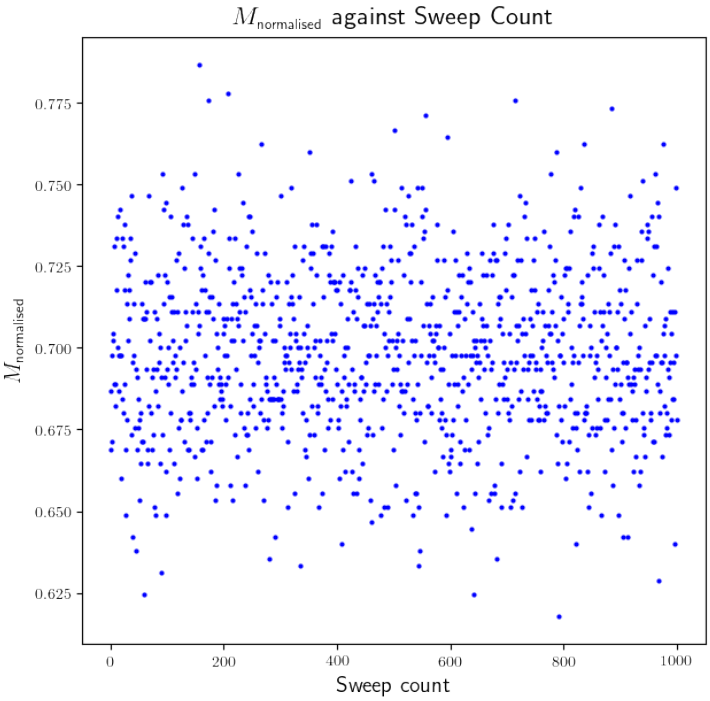

Figure 3: Scatter plot of the lattice's magnetisation (normalised) against the sweep cycle number for $J/k_BT = 0.1$ and $B/k_BT = 0.6$. Notice that the magnetisation centres around a positive value. This coheres with the idea that for values of B/k_BT which are large relative to values of J/k_BT, the magnetisation always converges to a positive value (i.e., most of the spins eventually attain parallel alignment by becoming 'spin-up'). This is to be expected because the effect of the external magnetic field now dominates and presides over the effect of the exchange interactions. The $-B\sum_i S_i$ term in equation (1) ensures that the energy E is minimised only if $\sum_i S_i > 0$ (for sufficiently large values of B/k_BT).

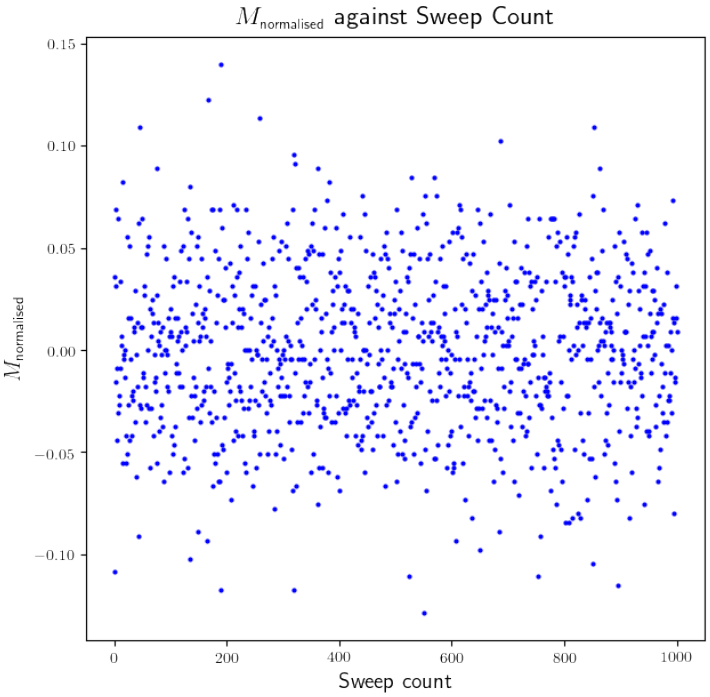

Figure 4: Scatter plot of the lattice's magnetisation (normalised) against the sweep cycle number for $J/k_BT = 0.1$ and $B/k_BT = 0$. Notice that the magnetisation centres around a negligibly small value close to 0. This coheres with the idea that the magnetisation converges to a low value (in terms of magnitude) for sufficiently low values of J/k_BT, which is consistent with Physics in two ways. A (sufficiently) low value of J/k_BT arises from two possible causes: either that J is small (i.e., the exchange interactions between spins are weak), or that $T > T_c$, where T_c is the Curie temperature of the lattice. In either case, the lattice would have effectively lost its ferromagnetism, which is consistent with the magnetisation converging to a low value (in terms of magnitude).

Main Plots

Following the sensibility of the preliminary data, more comprehensive data were produced. Relevant plots with accompanying analyses (where applicable) are provided in what follows. Some theoretical results were used to compare with the data produced. Onsager (1944) obtained analytic solutions for the critical value of J/k_BT at which phase transitions occur, in the absence of an external magnetic field (i.e., $B = 0$) — $(J/k_BT)_c = \frac{\ln(1+\sqrt{2})}{2} \approx 0.441$ — as well as the (stabilised) magnetisation when $J/k_BT > (J/k_BT)_c$ — $M_{\text{stable}} = [1 - \sinh^{-4}(2J/k_BT)]^{1/8}$.

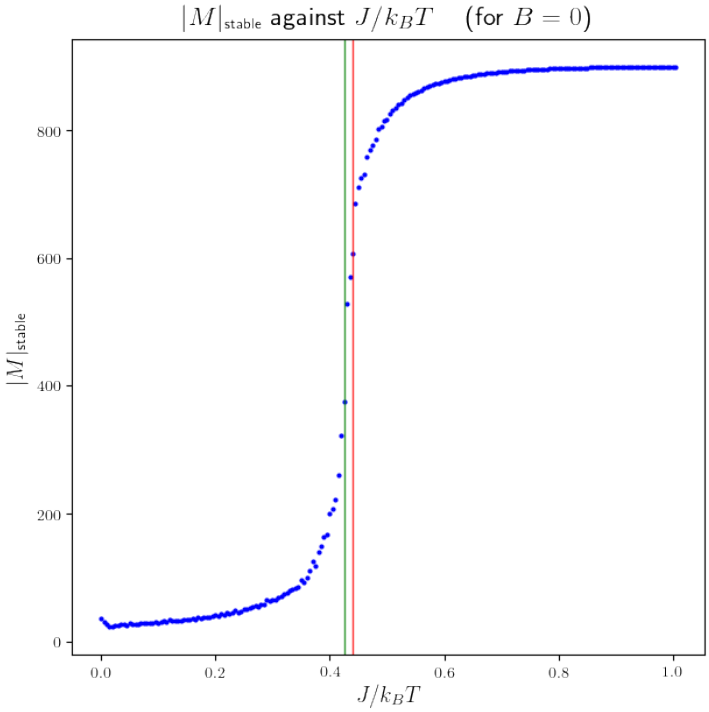

Figure 5: Scatter plot of the lattice's stabilised magnetisation* against J/k_BT, for $B/k_BT = 0$. The red, vertical line marks out Onsager's analytical solution $(J/k_BT)_c \approx 0.441$. The green, vertical line marks out the *estimated* critical value of J/k_BT according to the data, which was determined to be 0.425 via an algorithm that calculated the value of J/k_BT for which the change in $|M|_{\text{stable}}$ was the greatest. The two values are remarkably close to each other (within 4%).

*The values for these were obtained by collecting the magnetisation values from the 500^{th} sweep cycle onward for specified interval values of J/k_BT in the range $[0, 1]$, then computing the mean of the collected magnetisation values for each of the interval values of J/k_BT.

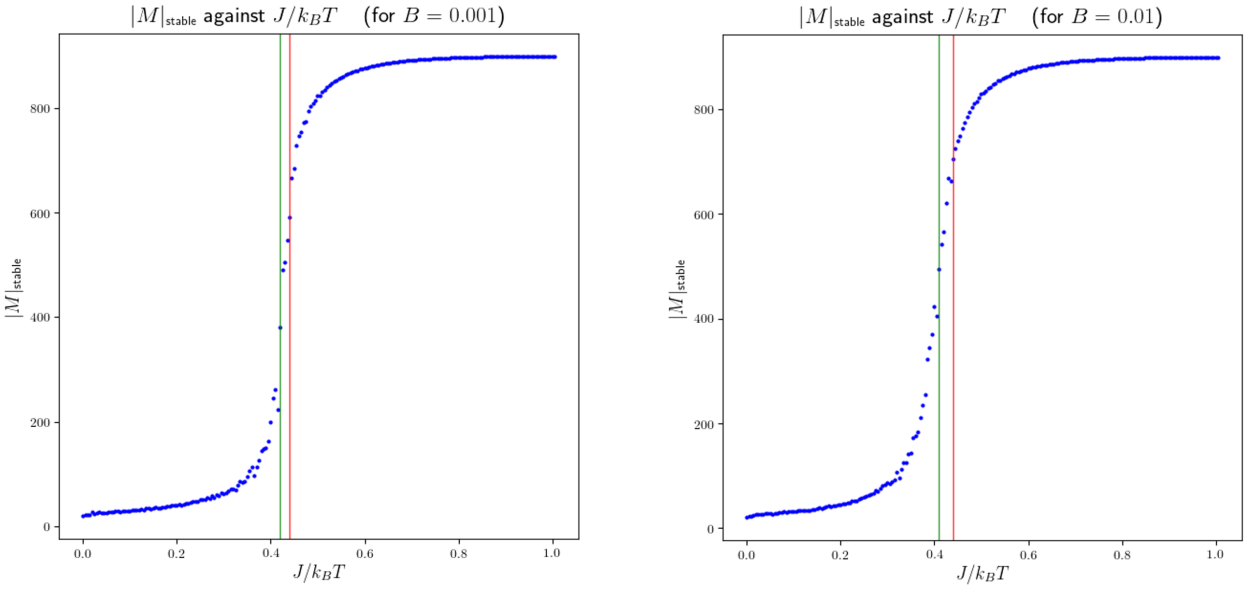

Figure 6: Scatter plot of the lattice's stabilised magnetisation against J/k_BT, for $B/k_BT = 0.001$ **(left)**. Scatter plot of the lattice's stabilised magnetisation against J/k_BT, for $B/k_BT = 0.01$ **(right)**. Just like in Figure 5, the red, vertical lines mark out Onsager's analytical solution $(J/k_BT)_c \approx 0.441$, and the green, vertical lines mark out the estimated critical values of J/k_BT according to the data, which were determined to be 0.420 for $B/k_BT = 0.001$ (left plot) and 0.410 for $B/k_BT = 0.01$ (right plot). Onsager's analytical solution is applicable only when $B = 0$, but it was nevertheless marked out in both plots to serve as a visual reference point for the estimated critical values. We can infer from Figures 5 and 6 that the (estimated) critical value of J/k_BT seems to decrease as B/k_BT increases. This is physically sensible because J and B fundamentally represent 'forces' (remember: J governs the strength of exchange interactions, B governs the strength of the external magnetic field) that compete with each other for influence over the spins.

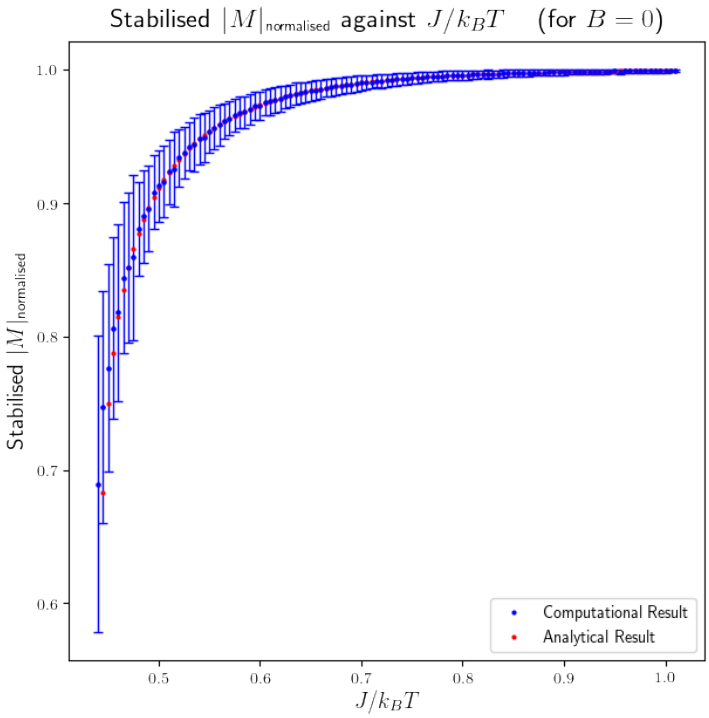

Figure 7: Scatter plots of the lattice's stabilised magnetisation (blue dots with error bars) against J/k_BT for $J/kBT > (J/kBT)_c \approx 0.441$ and Onsagers analytical solution for M_{stable} (red dots), where $B = 0$ in both cases. We can see that there is very good agreement between the computed data and the analytical solution, since the plotted points for the latter lie either within the error bars of, or very close to, the plotted points for the former.

Discussion

We have seen that and examined how the generated data are consistent with theoretical predictions in both the qualitative and quantitative senses. As such, we now shift our attention to outline why the Metropolis algorithm worked so well, how it might have caused problems, and what alternative algorithms could be implemented.

Why the Metropolis algorithm works

In the given context, the Metropolis algorithm works by causing the lattice to perform a *random walk* through the probability space of all accessible configurations and eventually settle into the region of highest density (i.e., the configuration of lowest energy). How does it do this? Imagine that the lattice is in some configuration $\mathbf{S}_t$. According to the acceptance ratio supplied by the Metropolis algorithm — $r = P(\mathbf{S}')/P(\mathbf{S}_t)$ — if the lattice attempts to transition to a configuration $\mathbf{S}'$ that is more probable than the current configuration $\mathbf{S}_t$ (i.e., in the probability space of configurations, $\mathbf{S}'$ is in a region of higher density than $\mathbf{S}_t$), then $r > 1$ and the transition will always be accepted. However, if the lattice attempts to transition to a configuration of lower probability, then $r < 1$ and the transition may not be accepted. In fact, the lower the probability of the proposed configuration, the less likely the transition would be accepted. Thus, from the 'big-picture' perspective, the lattice will *tend* to move to and stay in high-density regions of the probability space of configurations while only occasionally visiting the low-density regions [6]. This is why the Metropolis algorithm enables the lattice to reach the equilibrium state or configuration of lowest energy.

Limitations of the Metropolis algorithm

One issue (that is not unique to the algorithm itself) is simply that the distribution of the samples is necessarily an imperfect reflection of the actual distribution. Phrased in terms of the given context, the issue is that it is practically impossible for the lattice to visit every possible configuration in a finite time [7], which means that the distribution of sampled magnetisation values is necessarily an imperfect reflection of the distribution of all possible magnetisation

values[4]. This means that the computed value of the lattice's (stabilised) magnetisation would very likely differ from the actual value of the magnetisation.

The above issue would not be significant if the difference between the computed result and the actual result were very small. However, it can be amplified due to the Markovian property of the Metropolis algorithm, which brings us to the second issue: since the transition to a proposed state always depends on the prior state, the samples obtained would be *correlated* and thus *non-independent*. Hence, the disparity between the distribution of samples and the actual distribution would become even greater. Phrased in terms of the given context, the issue is that the lattice's transition to a proposed configuration always depends on the preceding configuration, so the configurations visited would tend to have similar probabilities (especially for a small number of sweep cycles). Consequently, the distribution of the sampled magnetisation values would likely be skewed and hence not be an accurate representation of the distribution of all possible magnetisation values. This means that the computed value of the lattice's (stabilised) magnetisation could differ *substantially* from the actual value of the magnetisation, especially for a low number of sweep cycles.[5]

An alternative to the Metropolis algorithm

Gibbs sampling is a widely used MCMC sampling method that can serve as an alternative to the Metropolis algorithm for simulating the Ising model [2]. An outline of how it would be implemented is as follows: (1) select a spin in the lattice at random, (2) compute the probability that the spin has a value of $+1$ given the local field E_f ★, (3) generate a random number x on the interval $[0,\ 1]$, (4) set the chosen spin's value to $+1$ if $P(+1|E_f) > x$ or -1 otherwise, and (5) repeat from step (1) while ensuring that the spin selected is different each time.

★ This probability is given by

$$P(+1|E_f) = \sigma(2E_f/k_BT) = \frac{1}{1 + e^{-2E_f/k_BT}}$$

where σ denotes the sigmoid function, and $E_f = J\sum_j S_j + B$ is the *local field* due to the chosen spin's neighbouring spins and the external magnetic field.

Conclusion

To conclude, the Ising model for ferromagnetism was simulated and studied for a two-dimensional square lattice via implementing the Metropolis algorithm. The qualitative and quantitative results of the study were found to be consistent with Physics and analytical results from the literature. The Metropolis algorithm was also examined and evaluated, with a potential alternative to it suggested for a possible extension of this study.

---

[4] Remember that the magnetisation value for a configuration is obtained by summing the spin values of the lattice in that configuration. In an $m \times m$ lattice, there would be 2^{m^2} possible configurations, and thus 2^{m^2} possible magnetisation values. This is because there would be m^2 spins in the lattice and each spin would have 2 possible values: $+1$ or -1.

[5] This is why a *burn-in* period was implemented, where the lattice in its initial configuration was sweeped across a certain number of times to avoid the effect of the initial configuration. It is also why the stabilised magnetisation, M_{stable}, was calculated from the magnetisation values collected after the 500^{th} sweep instead of the magnetisation values collected from all sweeps.

References

[1] Bennett, D. (2016). Numerical Solutions to the Ising Model using the Metropolis Algorithm. Retrieved from: https://www.maths.tcd.ie/~dbennett/js/ising.pdf.

[2] MacKay, D. J. (2003). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press.

[3] Kotze, J. (2008). *Introduction to Monte Carlo methods for an Ising Model of a Ferromagnet*. arXiv preprint arXiv:0803.0217.

[4] Department of Physics, University of Oxford. *CO28: Ferromagnetism*. Retrieved from: https://www-teaching.physics.ox.ac.uk/practical_course/scripts/srv/local/rscripts/trunk/Computing/CO28/CO28.pdf.

[5] Onsager, L. (1944). *Crystal statistics. I. A two-dimensional model with an order-disorder transition*. Physical Review, 65(3-4), 117.

[6] Roberts, G. O. (1996). *Markov chain concepts related to sampling algorithms*. Markov chain Monte Carlo in practice, 57, 45-58.

[7] Fricke, T. (2006). *Monte Carlo investigation of the Ising model*. Retrieved from: https://www.asc.ohio-state.edu/braaten.1/statphys/Ising_MatLab.pdf.

Appendix

The core modules used in the Python script — the lattice creator, the sweep function, and the simulator — are shown below.

The lattice creator

```
def create_lattice(m):
    '''
    Generates an m x m array, where each element is (pseudo)randomly assigned a value of either 1 or -1. The array is
    thus a simulation of an m x m lattice with a (pseudo)randomly determined configuration of +1 spins and -1 spins.
    '''
    lattice = np.random.choice([1,-1], size=(m,m))
    return lattice
```

The sweep function

```
def sweep(S, J_kT, B_kT):
    '''
    Outputs an m x m array representing the subsequent configuration of the spins after one 'sweep' across the
    inputted array S_init. A 'sweep' is the process of flipping or not flipping the spin of every array element,
    with the decision to flip or not to flip being based on the Metropolis-Hastings algorithm. J_kT = J/(k_B*T),
    and B_kT = B/(k_B*T).
    '''
    m = S.shape[0]
    list_randomrowindices = np.random.choice(m, m, replace=False).tolist()
    list_randomcolumnindices = np.random.choice(m, m, replace=False).tolist()

    for i in list_randomrowindices:
        for j in list_randomcolumnindices:
            s = S[i,j] #pick a random element from S_init (without replacement).
            neighbour_sum = S[i,(j+1)%m] + S[i,(j-1)%m] + S[(i+1)%m,j] + S[(i-1)%m,j] #compute sum of immediately
            #...neighbouring spins. The modulo operations help impose the periodic boundary conditions, which say that
            #...for any particular row/column in the array, its first element would be a neighbour of its last element,
            #...and vice versa. We can visualise how this works by imagining the array as a 2D sheet and folding it into
            #...a torus.
            delta_E_kT = J_kT*2*s*neighbour_sum + B_kT*2*s #compute change in (effective) energy. The factor of 2 exists
            #...because if s were flipped, the magnitude of the change in energy would be equal to twice the initial energy.
            if delta_E_kT <= 0:
                s *= -1
            elif np.random.rand() < np.exp(-delta_E_kT):
                s *= -1
            S[i,j] = s
    return S.copy()
```

The simulator

```
def run_simulation(S_0, J_kT, B_kT, sweep_total):
    '''
    Performs the simulation by sweeping across an initial lattice S_0 a number of times (number = sweep_total).
    Returns a list of relevant values (e.g., the stabilised magnetisation).
    '''
    M_list = []
    M_abs_list = []
    M_cumulative_list = []
    sweep_count_list = []
    S_colormap_list = []
    colormap_sweep_count_list = []


    # To avoid effect of initial state
    for count in range(50):
        S_init = sweep(S_0, J_kT, B_kT)


    # Start proper sweep
    for sweep_count in range(sweep_total + 1):
        S_next = sweep(S_init, J_kT, B_kT)
        S_init = S_next
        M = calc_magnetic_moment(S_next)
        M_list.append(M)
        M_cumulative_list.append(sum(M_list))
        M_abs = abs(M)
        M_abs_list.append(M_abs)
        sweep_count_list.append(sweep_count)

        if (sweep_count) % (sweep_total/8) == 0:
        # Extract S_next at 8 different stages to plot the color maps later
            S_colormap_list.append(S_next)
            colormap_sweep_count_list.append(sweep_count)


    # Obtain a list of |M| values from the 500th sweep till the last sweep. |M| is assumed to have reached a stable value by
    #...the 500th sweep; we make this supposition based on prior plots of |M| against sweep number for different values of
    #...J/k_BT and B/k_BT.
    stable_M_abs_histogram_data = M_abs_list[500:]

    # Fit a normal distribution to the list of values of |M|, and compute the mean and standard deviation.
    mu, std = norm.fit(stable_M_abs_histogram_data)

    return [mu, std, M_list, M_cumulative_list, M_abs_list, sweep_count_list, S_colormap_list, colormap_sweep_count_list]
```