CO28: Ferromagnetism

Alex McInerny, shil5991

May 6, 2024

1 Abstract

The 2D Ising model is numerically analysed in `MATLAB` to provide insight on how ferromagnetic atomic spins behaviour in conditions of changing temperature and external magnetic field. The metropolis-hastings algorithm is used, and periodic boundary conditions are employed to recreate conditions you'd find in a real-life material. An animation is produced to visualise in-time how the spins are changing. In the report, we confirm that at low temperatures, steady states are quickly reached, whereas at higher temperatures spins flip much more randomly, and at exceedingly high temperatures flipping is nearly random. We confirm experimentally that the critical temperature at which a phase shift occurs between the two states is at $J/k_BT_c \approx 0.44$ when no external magnetic field is applied. We also confirm that the presence of a magnetic field means spins are more likely to align in a particular direction, and that the critical temperature is increased in a way correlated with the strength of the field.

2 Introduction

Ferromagnetism is a phenomenon which arises due to a collection of atomic spins, aligned in a way such that their associated magnetic moments point in the same direction, yielding a resultant magnetic moment (magnetisation) that's macroscopic in size. Not many materials are able to exhibit this property, and they often contain (either in pure, alloy or compound form) iron, cobalt, nickel, or other rare earth metals [1]. This effect often found in every-day permanent magnets, and has extensive uses in many areas including electrical devices such as generators, magnetic storage and loudspeakers.

The two-dimensional Ising model is a simple theoretical description of ferromagnetism developed by Wilhelm Lenz in 1920, and was the focus of Ernest Ising's PhD thesis in 1925 [2]. It allows for easy macroscopic modelling of magnetisation in magnetic materials. The model consists of an array of discrete variables, representing the magnetic dipole moments of atomic spins, which take one of two values ($+1$ or -1). The lattice arrangements allows spins to interact with their local environments. Neighbouring spins which have the same value 'agree', and have lower energy than those which 'disagree'. The system tends towards the lowest energy state, however is disturbed by heat.

In the following report, we discuss in detail the theory underpinning the model, how it was implemented in `MATLAB`, consider the results when changing external parameters, and conclude our findings.

3 Theoretical Background

Consider N atoms arranged at the intersections of a square lattice, within the presence of a magnetic field of flux density $\mathbf{H} = H\hat{\mathbf{z}}$. Assume each atom is identical, and a spin-1/2 system. Therefore, each atom may have spin $S_i = +1$ (spin up) or $S_i = -1$ (spin down), where S_i is the $\hat{\mathbf{z}}$-component of the ith atoms atomic spin.

The total energy of such configuration is gven by [3]

$$E = -J\sum_{i=1}^{N}\left(\sum_{\text{neighbours } j} S_iS_j\right) + B\sum_i S_i, \tag{1}$$

where the first term is a sum over the nearest neighbours of i, i.e., the atoms above, below, right, and left of i, J is the exchange energy ($J > 0$ for ferromagnets), and $B = -\mu H$ is the magnetic field strenght, with μ being the atomic magnetic moment of the atoms in question.

Physically, the right summation of Eq.(1) represents the effect of the spin-spin interaction between neighbouring atoms. It is a statement that overall energy is decreased while neighbouring spins are aligned – a consequence of the Pauli Exclusion Principle. The left term is a summation over all atoms i, and gives the effect on energy due to the application of a uniform field. [4]

There are many options to calculate solutions to the Ising model, and it can even be analytically solved in 1 and 2 dimensions, however these solutions are not simple. One option is to use Mean Field Theory [4], which assumes thermal variation in the system is small and therefore negligible, allowing each particle to be considered to not interact with each other, but instead with the average field which encapsulates the behaviours of all particles in the vicinity, thus approximating a solution. This is also not easy to solve, so an instructive approach is using a Monte-Carlo based Metropolis-Hastings method to numerically solve the Ising model, allowing visualisation of all the spin states, and seeing how they change with temperature and external field variations. This approach is based on the following algorithm [3]:

1. Initialise an $N \times N$ lattice. Either every element is 1, or they're randomly chosen to be ± 1.
2. Randomly choose a site inside the lattice.
   (a) Calculate the system's energy via Eq.(1) for the current spin state (E_1), and the state if the node in question were to be flipped (E_2).
   (b) Calculate $\Delta E = E_2 - E_1$.
      i. if $\Delta E < 0$: Flip spin.
      ii. if $\Delta E \geq 0$: Flip spin with probability proportional to the Boltzmann factor $P_{\text{flip}} = \exp\left\{-\frac{\Delta E}{k_{\text{B}}T}\right\}$.
3. Repeat step 2 until a complete sweep of the lattice is complete (whereby each element is considered only once).
4. Complete as many sweeps as required until a stable equilibrium has been reached.

It is not necessary to compute the energy of the whole lattice twice when calculating the change in energy between after testing the spin-flip (this may be computationally expensive for large N). You can note most of the energies won't change in the sum, thus any contribution from non-neighbouring sites should cancel. Computing this gives, for spin test-flipped at the (i, j)th position

$$\Delta E = 2S_{i,j}\left[J\left(S_{i,j+1} + S_{i,j-1} + S_{i+1,j} + S_{i-1,j}\right) - B\right]. \tag{2}$$

This model clearly has one issue; at the edges of the lattice, there is no $(N + 1)$th (or -1th) spin. Evidently this should cause issues since, if you ignore edge-cases or padded out all the sides with say a random initially assigned spin that doesn't change, you will have inaccurate energy changes meaning spins would be inappropriately flipped. This can be overcome by using periodic boundary conditions [5], by which if you were at the (k, N)th spin, instead of taking $(k, N + 1)$ to be a randomly chosen value/ignored in the calculation, you take it to be the $(k, 1)$th spin. In this way, it is as though your sample infinitely repeats periodically in all the directions.

4 Implementation

A few functions were created to help solve the problem. These were each used within the main script, where you're able to adjust starting parameters. Implementation became the simple issue of converting the theory explained in the previous section to code. The functions created were split into two categories; those used to actually calculate quantities, and those for plotting data allowing for easy analysis.

4.1 `sweep` function

The brunt of the problem is iterating randomly through the array, and flipping spins dependant on the conditions in Itm.(2b). The `sweep` function takes inputs: the `spin` matrix, which just contains information about the current state of the system; and the parameters `N` (the size of the spin matrix), `J` (the exchange energy), `B` (the applied magnetic field strength), and `T` (the temperature of the system). The function returns the matrix `spin` after one complete sweep (i.e., after each site has been visited once, and updated).

In order to visit each site of the matrix once, the inbuilt function `randperm` is used. It is supplied with the number N^2, which is the number of sites in the lattice, and creates a random permutation of numbers from 1 to N^2, which, for this sweep, shall be the order that the sites of the grid are visited. This randomly permutated list is iterated over via. a for loop. Then, the `ind2sub` function converts the iterator into the corresponding spin matrix element index (`i` and `j`, corresponding to a position in an array).

Modular arithmetic is used to input periodic boundary conditions when computing the change of energy `DeltaE`. For example, to account the spin to the 'right' of the (i, j)th spin (which may be out of bounds if $j = N$, since $j + 1 = N + 1$ is not within the matrix), you replace the coordinates $(i, j + 1)$ with $(i, j \pmod N) + 1)$, which employs the periodic boundary conditions mentioned in the previous section. This was done using the inbuilt `mod` function in `MATLAB`. Similar expressions were used for the remaining points.

Lastly, an if statement was used to determine whether the spin ought to be flipped; if the change in energy was less than zero, or a uniformly generated random number in the interval $[0, 1]$)generated via. `rand` function) was less than or equal too the Boltzmann factor $e^{-\Delta E/T}$, the spin was flipped. Else, the spin remained in it's initial state, and the program continued.

This procedure was completed for each value in the random permutation, meaning each lattice site was visited once in the previously mentioned random order. Once the for loop was completed, `spin` (the matrix containing info about the spins) was returned.

4.2 `CO28_ferromagnetism` function (`main`)

Once `sweep` is made, all that is left to find how the spins change is to set the initial conditions, and iterate the sweep function the desired number of times, while extracting the information required. This is all done in the main.

After setting all desired quantities such as J/k_BT, B/k_BT, N, and the number of sweeps, a spin matrix is initialised. Each element of this $N \times N$ 2D-array is a random number of either 1 or -1 (pertaining to spin up and down respectively), which is calculated by raising -1 to the power of a random matrix containing 1 or 2. It is also possible to initialise the matrix containing only 1 or -1.

A for loop then iterates over the desired number of sweeps. In this for loop, the spin matrix is updated such that the result of applying sweep on the previous spin matrix is placed in the next element of the variable '`spins`', which is a 3D-array. Furthermore, a vector containing the total magnetic moment , '`M`', which just sums up the spins array for that iteration number, and also the cumulative magnetic moment , '`cumulativeM`', is calculated by summing up all previous values of `M`. Once the desired number of sweeps is reached, all plotting functions are called.

4.3 `plot_JkTList_vs_averageAbsM` function

This function is a slight adaptation for the iteration in the main function. It allows for two more nested for loops; one iterates through a list of values for J/k_BT (but allows for the start/end positions, and the number of temperatures to be chosen freely) meaning the magnetisation's can be calculated for a series of temperatures with the same initial spin-matrices and probabilities (to see how magnetisation varies with temperature), and the other averages these readings over a desired number of averages. This final averaging is required since the random behaviour means you don't see the full picture over one set of sweeps– there are often outliers due to the randomness of the initial configurations– taking an average reduces that and allows for smooth curves to be drawn when plotting temperature verses magnetisation.

4.4 `plot_heatmap` function

`plot_heatmap` takes the spins matrix, containing information about the spin matrix for every iterations produced in the main function, and plots the desired number of heatmaps of the spins at equally spaced intervals. For example, in the following script if there were 80 sweeps and you asked for 9 graphs to be displayed, it should show frames (1, 11, 21, ..., 81). This allowed you to analyse visually if equilibrium had been reached, and the path taken to get there.

In order to do this, the function is passed the spin matrix, and a tiled layout is started. A for loop then iterates over the number of graphs you wish to display, and creates a tile for each frame. A heat map of that frame is plotted using the `heatmap` function, and it's labeled with the frame number. Then the next tile is started with another frame plotted, etc.

4.5 `plot_bar` function

Plotting a bar graph of the magnetisation allows you to see how/if equilibrium was reached.

The function gets passed both the magnetisation and cumulative magnetisation vectors and just plots a bar graph of them against sweep number again in a tiled layout.

4.6 `plot_animation` function

To visualise what was happening, an animation was used to compare how spins were changing between one (or multiple) sweeps.

This was implemented using the `movie` function. Because a graphing routine takes a lot of time, it was decided to not print every frame, but to print every nth frame instead. The code just iterated through the frames which were to be included in the gif, graphed them, and stored the frame using the '`getframe`' function. Once all frames had been graphed, calling `movie` on the structure containing the frames meant they were played in quick succession. The code for this can be viewed in Appx.(A.6).

5 Analysis and Results

Results were computed for a series of values of temperature, and magnetic field. Throughout, a 30×30 grid is used.

5.1 Ranges of temperatures with $B = 0$

Initially, $B = 0$ was constrained to analyse the results without the presence of a magnetic field. This allowed us to see how temperature alone effected ferromagnetic materials.

At low temperatures (Fig.(1), which related to high inverse temperature J/k_BT), equilibrium is very quickly reached. Looking at the top right graph of Fig.(1), notice that the graph flattens quickly (within around 15 sweeps), and then is non-changing after around 65 iterations. This is backed up by the spin frames, where you can see, having started with a random initial configuration, the spins quickly start to align, and then fully reach a steady state.

Note that the final spin configuration may vary; it's fully dependant on the random initial conditions and order in which the sweep is carried out. It's possible every 'atom' could be spin up, down, or have reached some equilibrium with, say, a bar of spin up surrounded by spin down as in the figure, etc.

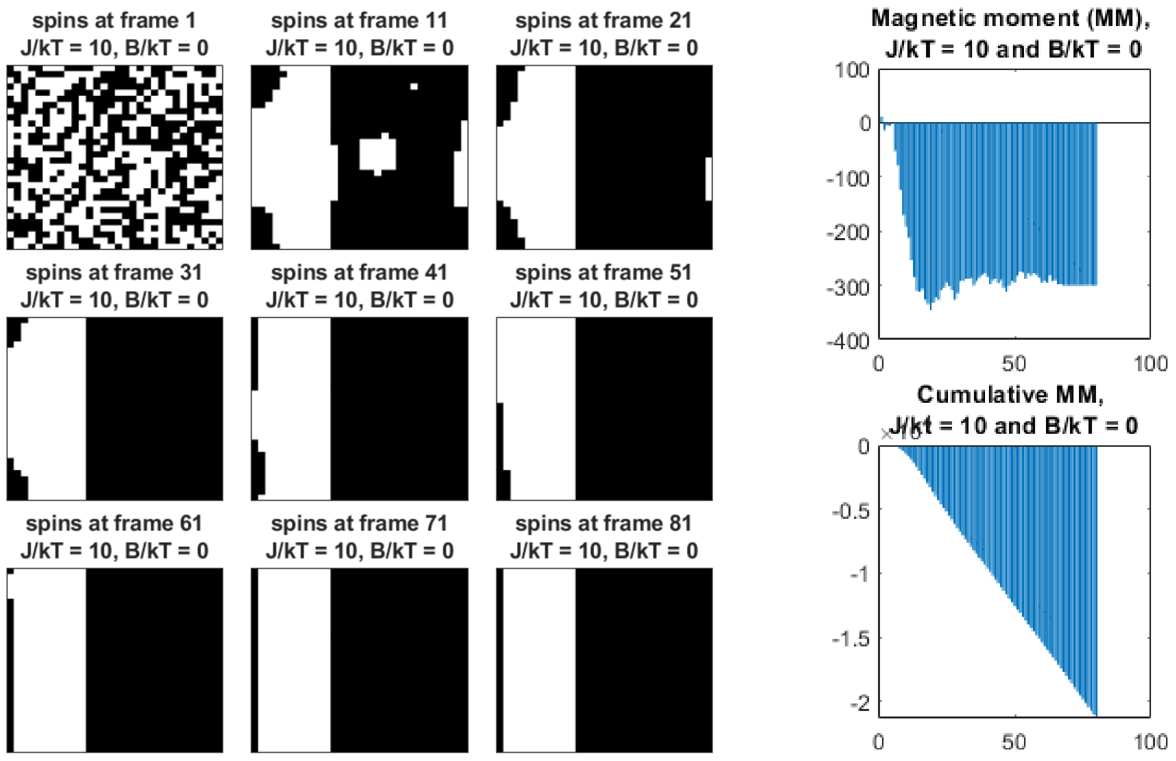


Figure 1: Low temperatures, no external magnetic field.

At a temperature around the critical point of $J/k_BT \approx 0.44$, a steady state is never fully reached (see Fig.(2)). Although you can see by the positive gradient of the cumulative magnetic moment graph that the atoms are tending to 'white', as in spin up, there are often large clumps of spin down, or just singular ones which appear randomly. This comes as no surprise, as at a higher temperature, the probability of a spin flipping is increased regardless of whether it is in a favourable environment or not, so lone spin flips (single black pixels) are likely to appear, and a chance few of these may cause the spin down area to start dominating, adding to the lacking stability of the system, as is seeming to happen in the final frame.

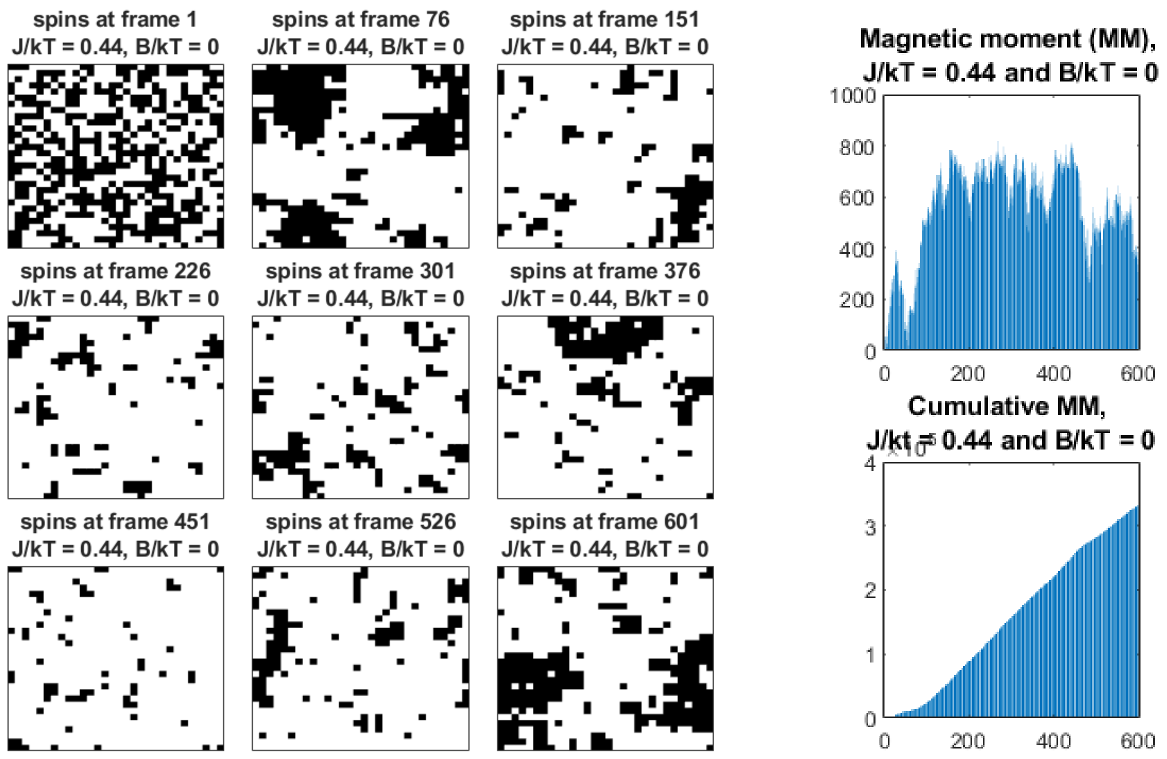


Figure 2: Critical temperature, no external magnetic field.

At high temperatures the spins flip much more randomly, as physically expected. After 600 iterations, equilibrium has clearly not been reached, and it doesn't look as though a steady state is possible– that's evident from the top right graph of Fig.(3)– each sweep there is a dramatic and seemingly random shift if the total magnetic moment, whereas at temperatures close to and below the critical temperature, the magnetic moment changes quite slowly. In the frames, you can see some minor clumping, but nothing to suggest there is a pattern to the behaviour of the spins.

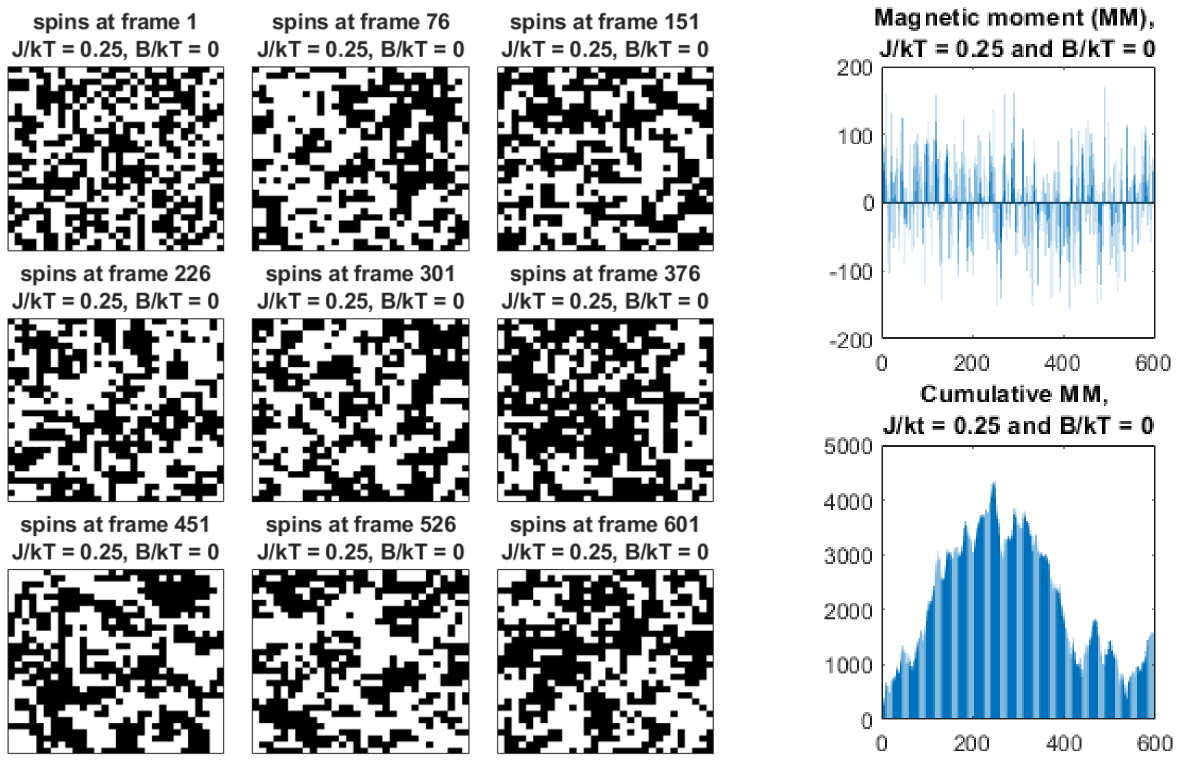


Figure 3: High temperatures, no external magnetic field.

The effect of how changing temperature affects magnetic moment is best seen when directly comparing the two. Fig.(4) clearly shows that, at low temperatures (the right side of the graph), the average absolute magnitude of magnetisation is nearly 900 (which would be the maximum), i.e., nearly all of the spins are pointing in the same direction. The fact it's closer to 700 is due to the fact you can reach equilibrium with not every spin facing the same direction, somewhat like you can see in the final frame of Fig.(1), so when you average the result turns out slightly below. As you approach the critical temperature of $J/k_BT_c \approx 0.44$, you can see a steep gradient. This represents a 'phase change' between ordered and disordered phases. At high temperatures (the left side), the average magnitude of magnetisation is very small (nearly zero). This is because the spins are changing fully randomly and no steady state is reached, so the spins fluctuate between up and down, so their average is 0.

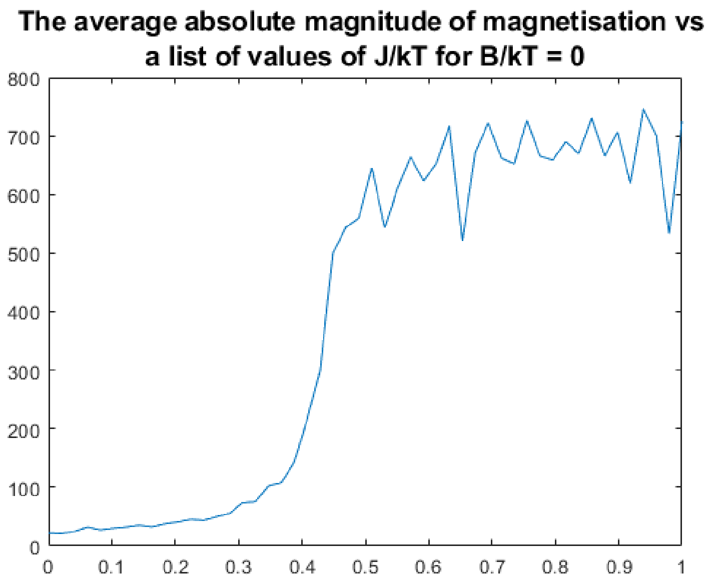
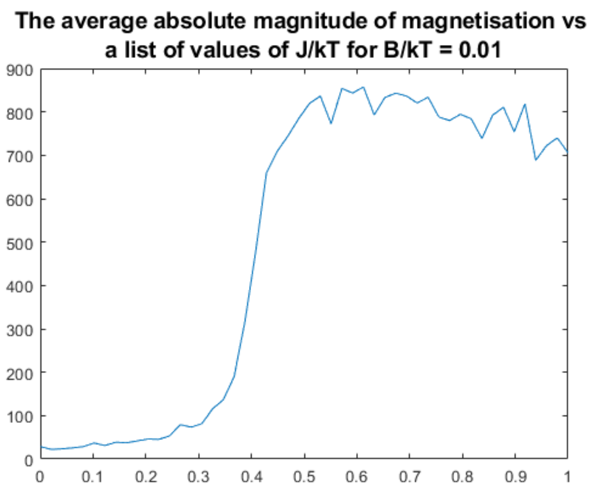


Figure 4: Absolute magnitude of magnetisation vs a list of 50 values of $J/k_BT \in [0, 1]$, averaged over 50 repeats.

5.2 Ranges of temperatures with $B \neq 0$

When a magnetic field is introduced, the atomic spins start aligning with the direction of the field, as is expected from the theoretical background section. The following figures contain graphs of the average magnitude of magnetisation against inverse temperature for different magnetic field strengths. They show that as field strength increases, the critical temperature (at which phase change occurs) increases (as in, a higher temperature is required in order to retain the random nature of the spins). They also show the unsuprising result that as the field increases, at low temperatures (when spins generally align with each other), the total magnetisation is nearly 900, as in all of the spins are aligned in the same direction as the **B** field. You even see, in the presence of the stronger field, as the temperature approaches infinity, the magnetisation is around 100, whereas if it were random as we saw with zero field, you should expect there the average absolute magnetisation to be zero.



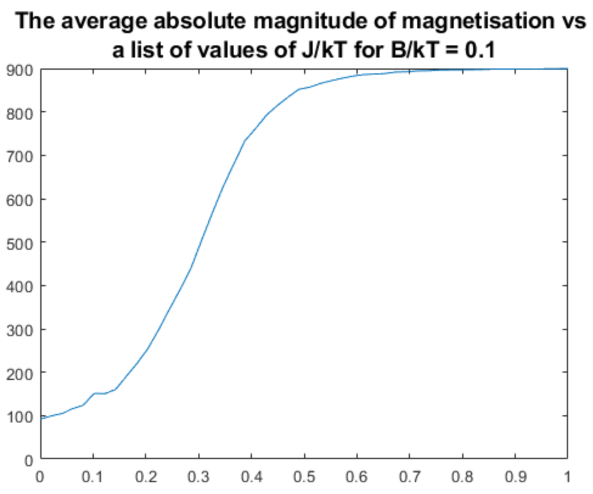


Figure 5: Graphs showing how external magnetic field affects the critical temperature and allignment of spins.

6 Conclusion

In this experiment, we explored the 2D ising model via. numerical approximation in `MATLAB`, which provided insight into the behaviour of ferromagnetic materials under changing temperature and external magnetic field. We derived by how much the energy of a configuration changed when a single spin was flipped, and applied this with periodic boundary conditions to a 30×30 grid, checking/flipping each element of the grid in a random order each sweep. Once the desired number of sweeps were executed, we analysed the configurations of the spins and magnetisation of the systems.

At low temperatures, the system rapidly reaches equilibrium, with spins alligned predominantly in one direction. If there was no magnetic field applied, then the direction was random, and often regions of spins were formed, whereas if there was a magnetic field, the spins aligned with it. At high temperatures the spins exhibit random fluctuations, preventing the attainment of a steady state. The critical temperature, which was given by $J/k_BT_c \approx 0.44$, marked a phase transition between ordered and disordered phases. Furthermore, the introduction of the external field had a large impact on spin alignment. Increasing the field strength had the effect of; at low temperatures aligning more spins, and at high temperatures increasing the average magnetic moment. Increasing field strength not only raised the critical temperature, but also increased the width of the phase transition so it happened over a broader range of temperatures.

The algorithm used was designed to be very general (allowing for different temperatures, field strengths, grid sizes, sweep numbers, and other parameters to be chosen arbitrarily), but also very visual and easy to use. A function was designed to create a GIF showing how the spins changed each iteration, which allowed for the sweep-by-sweep changes in spins to be easily observed, and other plots produced help gain a level of understanding which may not have been obtained if only the mathematical theory behind ferromagnetic materials was examined.

Although there are other methods which may be used to analytically solve the 2D problem, the method used throughout this paper allowed for a very heuristic approach to be taken, meaning it was a good exercise to learn the physical intuition behind the problem, and help connect electromagnetism, quantum theory, and statistical mechanics into condensed matter physics in a way which may not have been obvious to someone not having studied the subject before.

References

[1] Posted By: Aniziol. A comprehensive list of ferromagnetic materials -, Oct 2023. URL `https://www.moleymagneticsinc.com/a-comprehensive-list-of-ferromagnetic-materials/`.

[2] Apr 2024. URL `https://en.wikipedia.org/wiki/Ising_model`.

[3] URL `https://www-teaching.physics.ox.ac.uk/practical_course/scripts/`.

[4] URL `https://farside.ph.utexas.edu/teaching/329/lectures/node110.html`.

[5] Mar 2024. URL `https://en.wikipedia.org/wiki/Periodic_boundary_conditions#:~:text=Periodic%20boundary%20conditions%20(PBCs)%20are,computer%20simulations%20and%20mathematical%20models`.

A Appendix

A.1 Code for CO28_ferromagnetism function (main)

```
%     Author: Alex McInerny, Date: 05/05/2024
%     The main function of my ferromagnetism project; it is designed to
%     initialise all vairables, iterate a certain number of times, applying
%     the sweep function, storing required information, then call of the
%     required graphing funcitons.
clc
clear variables

%initialise constants
N = 30;
BkT = 0.01;
JkT = 10;
numSweeps = 200;

%initialise spin array.
spins(:,:,1) = (-1).^randi(2,N,N);
% spinInitial = ones(N,N);

%%
%iterate through for the required number of sweeps. Calculate the new spin
%matrix and also M and cumulativeM, and store them in vectors.
for i0 = 1:numSweeps
    spins(:,:,i0+1) = sweep(spins(:,:,i0), N, JkT, BkT);
    M(i0) = sum(spins(:,:,i0), "all");
    cumulativeM(i0) = sum(M, 'all');
end

%required plots.
plot_animation(spins)
numGraphs = 9;
plot_heatmap(spins, numGraphs, JkT, BkT)
plot_bar(M, cumulativeM, JkT, BkT)
%%

%plot of averageAbsM against JkT. In it's own section so it can be ran
%separately (as it may take time).
plot_JkTList_vs_averageAbsM(N, BkT, numSweeps, spins(:,:,1))
```

A.2 Code for sweep function

```
function [spin] = sweep(spin, N, JkT, BkT)
%     Author: Alex McInerny, Date: 05/05/2024
%     This function randomly sweeps through the spins array, calculates the
%     change of energy for each point, and flips the spin if it reaches the
%     dresired condition. After sweeping the entire grid (visiting each
%     point in a random order one time), it returns the swept matrix.
%
%     Inputs:
%     * spin: initial matrix of spins.
%     * N: Size of the spin matrix.
%     * JkT, BkT: constants.
%
%     Returns:
%     * spin: matrix containing the spins after a full sweep.

    for i1 = randperm(N*N)%start iterating through the random list of numbers

        [i,j] = ind2sub(size(spin), i1);%convert number to inxed.

        DeltaEkT = 2 * spin(i,j) * ( JkT * ( ... %calcualte change in energy
            spin(mod(i-2,N)+1, j) + ... above
            spin(mod(i,N)+1, j) + ... below
            spin(i, mod(j-2,N)+1) + ... left
            spin(i, mod(j,N)+1) ) - ... right
            BkT );
        %work out if spin should be flipped, and do it if required.
        if or( DeltaEkT <= 0, rand() <= exp( - DeltaEkT) )
            spin(i,j) = - spin(i,j);

        end

    end

end
```

A.3 Code for plot_bar function

```
function plot_bar(M, cumulativeM, JkT, BkT)
%     Author: Alex McInerny, Date: 05/05/2024
%     This function plots bar graphs of the magnetic moment, and cumulative
%     magnetic moment of a spin matrix.
%
%     Inputs:
%     * M: vector containing the magnetic moments of a spin matrix for
%       every iteration.
%     * cumulativeM: matrix containing the sum of all the previous magnetic
%     moments, updated each iteration.
%     * JkT, BkT: constants.

    figure;
    tiledlayout(2,1, "TileSpacing","tight", "Padding","compact");

    nexttile
    h = bar(M);
    title({'Magnetic moment (MM), ' ...
        sprintf('J/kT = %g and B/kT = %g', JkT, BkT)},'FontSize', 10)
    axis square
    h.EdgeColor = 'none';

    nexttile
    h = bar(cumulativeM);
    title({'Cumulative MM, ' ...
        sprintf('J/kt = %g and B/kT = %g', JkT, BkT)},'FontSize', 10)
    axis square
    h.EdgeColor = 'none';

end
```

A.4 Code for plot_animation function

```
function plot_animation(spins)
%     Author: Alex McInerny, Date: 05/05/2024
%     This function plots an animation of all the spins, to see how they're
%     changing.
%
%     Inputs:
%     * spins: initial matrix of spins. Contains all the historical values
%     for it aswel.

    %initialise constants
    N = size(spins, 3);
    frame_dist = 10;

    F(N) = struct('cdata',[],'colormap',[]); %create structure.

    cmap = [[0,0,0]; [1,1,1]]; % white [1,1,1] is spin up
    for j = 1:floor(N/frame_dist) %iterate through the required frames and plot them.
        h = heatmap(spins(:,:,j*frame_dist), 'Colormap', cmap ,'CellLabelColor','none');
        h.XDisplayLabels = nan(size(h.XDisplayData)); h.YDisplayLabels = nan(size(h.YDisplayData));
        set(gcf, 'Position',  [100, 100, 750, 750]);
        clim([-1 1]);
        colorbar('off')
        grid off;
        drawnow;
        F(j) = getframe(gcf);
    end

    %print gif.
    fig = figure;
    set(gcf, 'Position',  [100, 100, 750, 750]);
    movie(fig,F,1)
end
```

A.5 Code forplot_heatmap function

```
function plot_heatmap(spins, numGraphs, JkT, BkT)
%     Author: Alex McInerny, Date: 05/05/2024
%     This function plots an animation of all the spins, to see how they're
%     changing.
%
%     Inputs:
%     * spins: initial matrix of spins. Contains all the historical values
%     for it aswel.
%     * numGraphs: The number of graphs you'd like in the tyled layout
%     (i.e., how many frames to be viewed)
%     * JkT, BkT: constants

    tiledlayout("flow","TileSpacing","compact", "Padding","compact")

    cmap = [[0,0,0]; [1,1,1]]; % white [1,1,1] is spin up


    for i1 = floor(linspace(1,length(spins), numGraphs)) %iterate through number of
            %graphs wanted, and plot that specfic frame. done in tiled layout.
        nexttile
        h = heatmap(spins(:,:,i1), 'Colormap', cmap ,'CellLabelColor','none');
        h.XDisplayLabels = nan(size(h.XDisplayData)); h.YDisplayLabels = nan(size(h.YDisplayData));
        title({sprintf('spins at frame %g', i1), sprintf('J/kT = %g, B/kT = %g', JkT, BkT)})
        clim([-1 1]);
        colorbar('off')
        grid off;
        h.FontSize = 14;
        drawnow;
    end

end
```

A.6 Code for plot_JkTList_vs_averageAbsM function

```
    function plot_JkTList_vs_averageAbsM(N, BkT, numIterations, spins)
%     Author: Alex McInerny, Date: 05/05/2024
%     This function plots an animation of all the spins, to see how they're
%     changing.
%
%     Inputs:
%     * spins: initial matrix of spins. Contains all the historical values
%     for it aswel.
%     * numIterations: Number of sweeps completed in main.
%     * N: Size of the spin matrix.
%     * BkT: constant

    %initialise constants
    numAverage = 50;
    numTemperatures = 50;
    JkTList = linspace(0,1,numTemperatures);

    for i3 = 1:numAverage %iterate over averaging procdure
```

```
        for i2 = 1:numTemperatures %iterate through the list of temperatures
            for i1 = 1:numIterations %same procedure as in main, but with more dim arrays
                spins(:,:,i1+1) = sweep(spins(:,:,i1), N, JkTList(i2), BkT);
                M(i1, i2, i3) = sum(spins(:,:,i1), "all");
                cumulativeM(i1, i2, i3) = sum(M, 'all');
            end
            averageAbsM(i2) = mean(abs(M(numIterations, i2, :)));
        end
    end

    %plot graphs!
    figure
    plot(JkTList, averageAbsM)
    title({'The average absolute magnitude of magnetisation vs ' ...
        sprintf('a list of values of J/kT for B/kT = %g', BkT)}, 'FontSize', 14)
    xlim([0 10])
    ylim([-0.4 0.8])
end
```