CO28: Investigating Ferromagnetism using the Ising Model

Parkin Pham (orie4903)

February 19, 2025

1 Abstract

We investigate the behaviour of ferromagnets by simulating the simplified case of the 2D Ising model using the Metropolis-Hastings algorithm. By plotting the average values at equilibrium, we show there is a sharp transition in magnetic moment and a spike in heat capacity. This suggests the existence of a phase transition where above a critical temperature the system remains disordered with a random alignment of spins, behaving like a paramagnetic material, but below a critical temperature the system rapidly becomes ordered, with all spins aligning in the same direction thus giving the system a net magnetic moment.

2 Introduction

Atoms with unpaired electrons will have an overall magnetic dipole moment, which will align with an applied magnetic field, causing materials to gain a magnetic moment proportional to the magnetic field, which is known as paramagnetism. However in ferromagnetic materials, it is favourable for the magnetic dipoles of atoms to align with neighbouring magnetic dipoles. This allows for small domains of the material to have a majority of dipoles being aligned in one direction, thus giving the region a net magnetic moment. If a magnetic field is applied to the material, each individual domain will align with the field, and so the material will gain a net magnetic moment that persists even when the magnetic field is removed. It is also observed that at high temperatures, random thermal motions will break the alignment of dipoles causing the material to lose this magnetic moment and become paramagnetic. This occurs at a precise temperature for different materials, called the Curie point, where the magnetic moment suddenly drops to 0, which is indicative of a second order phase transition. [1]

We can investigate a simplified model of this behaviour, known as the Ising model. We consider a 2-dimensional $N \times N$ periodic lattice of spins which can have values $S_i = \pm 1$, and which is placed in a thermal bath of constant temperature. Considering the energy from the alignment between neighbouring spins and the alignment of spins with an external magnetic field, we can construct the following Hamiltonian, noting that the first term sums over adjacent pairs of spins, and the second term sums over all spins.

$$\hat{H} = -J \sum_{\langle i,j \rangle} S_i S_j + B \sum_i S_i \tag{1}$$

J is the exchange constant, such that when $J > 0$ it is favourable for neighbouring spins to be aligned, which causes ferromagnetism, and when $J < 0$, it is favourable for neighbouring spins to be anti-aligned, which causes antiferromagnetism. We note that the system has the least energy when the spins are aligned and opposite to the applied magnetic field, and so this must be the most energetically favourable configuration. However this macrostate corresponds to only one microstate, and so is entropically unfavourable compared to the macrostate with half the spins as 1 and half the spins as -1, which corresponds to the largest number of microstates. At equilibrium for a constant temperature, we expect the Helmholtz function $F = U - TS$ to be minimised, so at low temperatures we move towards the energetically favourable state with aligned spins, creating a net magnetic moment analogous to ferromagnetism. At high temperatures we move towards the entropically favourable case with random spins, with the net magnetic moment proportional to the applied magnetic field, corresponding to paramagnetism. It has been shown that for the case $B = 0$ there exists a critical temperature T_c given by the following. [2]

$$\frac{J}{k_B T_c} = \frac{\ln(1+\sqrt{2})}{2} \approx 0.4406867935 \tag{2}$$

This corresponds to a phase transition where for $T > T_c$ the net magnetic moment suddenly drops to 0, and the heat capacity and magnetic susceptibility sharply peak around T_c. From statistical mechanics, we can calculate the heat capacity from the variance in energy which will allow us to spot the phase transition. [3]

$$C = \frac{d\langle E \rangle}{dT} = \frac{\langle E^2 \rangle - \langle E \rangle^2}{k_B T^2} \tag{3}$$

We will also require the Gibbs distribution for the probability of a particular microstate α.

$$p_\alpha = \frac{e^{-\beta E_\alpha}}{Z(\beta)} \tag{4}$$

Where $Z(\beta) = \sum_\alpha e^{-\beta E_\alpha}$ is the partition function and $\beta = \frac{1}{k_B T}$.

3 Method

We can simulate the Ising model using the Metropolis-Hastings algorithm, which is a Markov chain Monte Carlo method that will simulate a random path through a space of possible states that will converge to a desired probability distribution. [4] [5] For the Ising model the algorithm goes through the following steps.

1. Start with an intial random configuration of spins.
2. Consider a new proposal configuration ν with one spin flipped.
3. Calculate the *acceptance ratio* of the probability of the new state to the current state μ, which is:

$$A = \frac{p_\nu}{p_\mu} = \exp\left(-\frac{E_\nu - E_\mu}{k_B T}\right) \tag{5}$$

   We can calculate $E_\nu - E_\mu$ very simply as it only depends on the spin being flipped and its neighbours.

$$E_\nu - E_\mu = 2J \sum_j S_i S_j - 2BS_i \tag{6}$$

   Where S_i is the spin being flipped and S_j are the spins of the 4 neighbours.
4. If $A > 1$, then change to the new configuration. If $0 < A < 1$, change to the new configuration with probability A.
5. Repeat steps 2-4 until the system has reached a stable equilibrium state.

When considering a new proposal configuration in step 2, we will iteratively flip each spin of the lattice in turn, with 1 "sweep" denoting when we have made one pass of the lattice. Once the system has reached equilibrium, we can take average values of energy, magnetic moment, and heat capacity.

The Metropolis algorithm allows us to randomly sample a non-trivial probability distribution $\pi(x)$, by constructing a Markov chain that converges to the stationary distribution $\pi(x)$. A Markov chain describes the probabilities of transitioning to any given state y from a given state x as $q(y|x)$. If our stationary distribution exists and is $\pi(x)$, we require the property of *detailed balance* is upheld, which means the probability of being in state x and transitioning to state y is the same as the probability of being in state y and transitioning to state x.

$$\pi(x)q(y|x) = \pi(y)q(x|y) \tag{7}$$

The Metropolis algorithm satisfies this condition by defining the acceptance ratio as:

$$A(x, y) = \min\left(1, \frac{\pi(x)}{\pi(y)}\right) \tag{8}$$

If the proposal distribution $g(x|y)$ (the probability of proposing a state x given a state y) is symmetric then since the transition probabilities are proportional to the acceptance ratios, we have that

$$\frac{q(x|y)}{q(y|x)} = \frac{A(x, y)g(x|y)}{A(y, x)g(y|x)} = \frac{\pi(x)}{\pi(y)} \tag{9}$$

Either $A(x, y)$ or $A(y, x)$ must be 1 by symmetry. Therefore the Metropolis algorithm satisfied detailed balance, which means after enough steps, our simulation will approach the probability distribution of the system at equilibrium.

The Metropolis is useful because it allows us to sample high-dimensional or otherwise intractable distributions by evaluating the value of any function proportional to the distribution, which is computationally much simpler. However the disadvantages are that the algorithm will require a *burn in* period of time steps before convergence is reached and values can be sampled, particularly near the phase transition, where the algorithm experiences a critical slowdown. The Metropolis algorithm also exhibits autocorrelation, which means successive samples in the Markov chain are correlated with each other, which reduces the statistical effectiveness of results.

4 Results

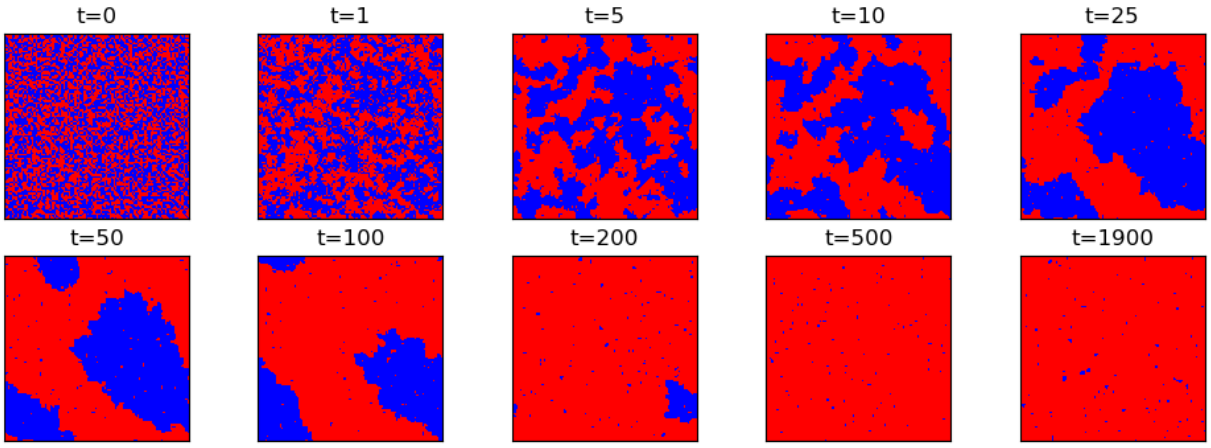

Figure 1: Snapshots of a 100x100 system below the critical temperature at various iterations of the simulation

Above in Figure 1 are snapshots of the Ising model with $\frac{J}{k_B T} = 0.6$, $B = 0$, at different sweep iterations for a 100x100 lattice, coloured by the spin at each grid point. Initially we set the system to a random configuration of spins, but very quickly within just a few sweep iterations the system becomes locally ordered, with spins aligning to create small regions with the same spin. Qualitatively, these regions slowly grow and merge until one region dominates and the system reaches an equilibrium with all spins aligned except for a few random thermal fluctuations, behaving similarly to a ferromagnetic system below the Curie temperature. We also note that even though the Hamiltonian is symmetric to global inversions of spin, the system spontaneously breaks this symmetry, gaining a net positive or negative magnetic moment with a 50% probability.

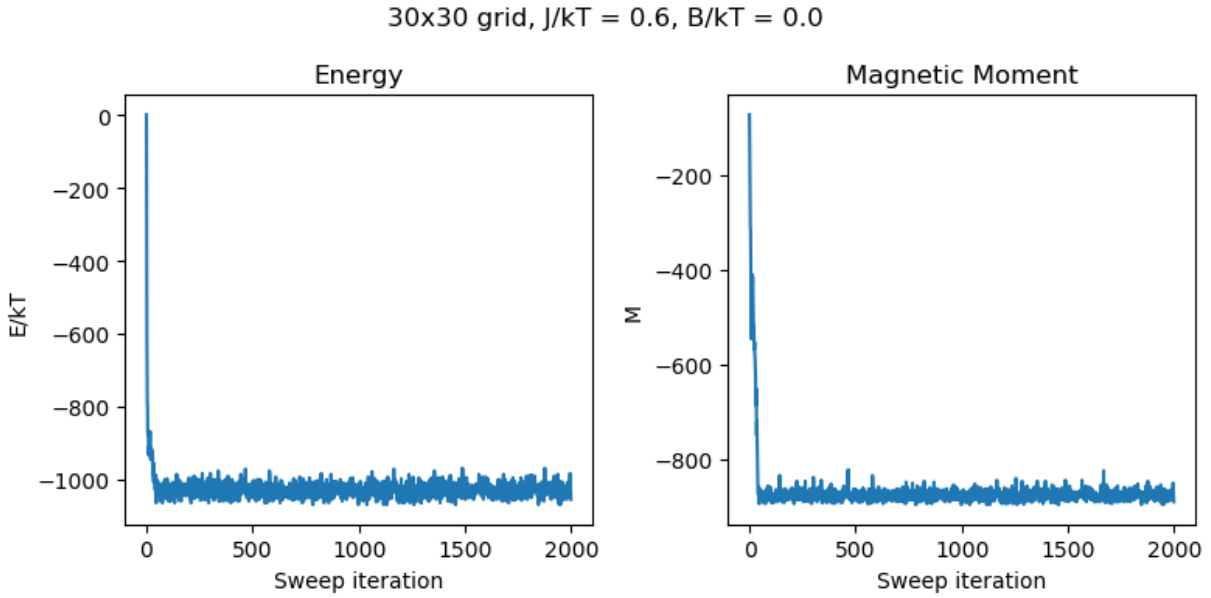

Figure 2: Graphs of the total energy and magnetic moment below the critical temperature

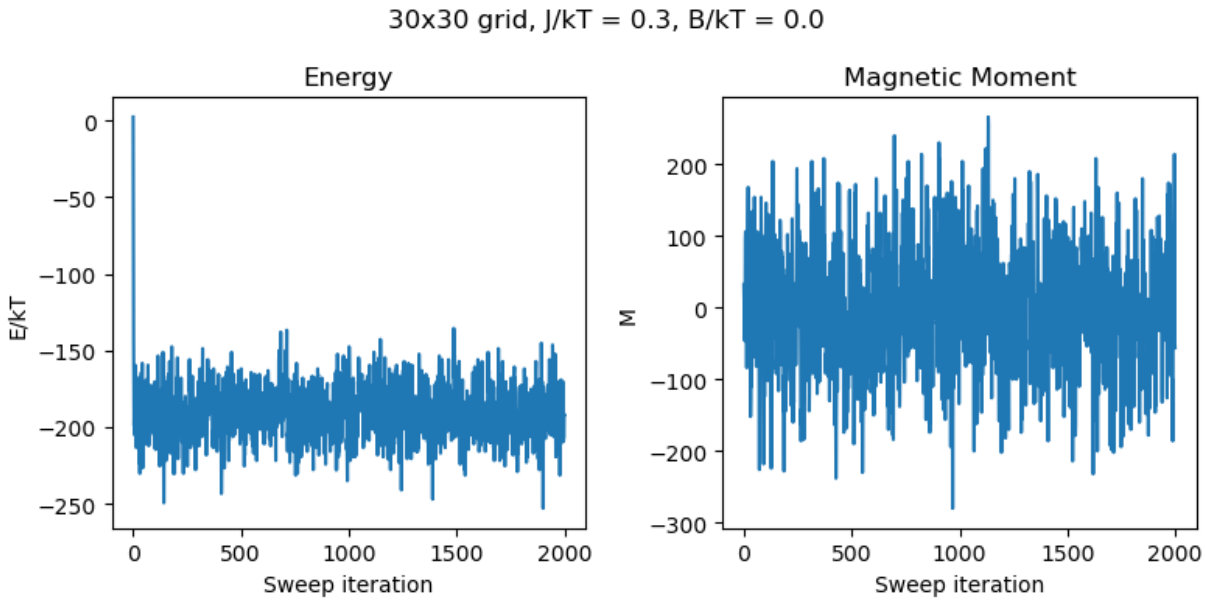

Figure 3: Graphs of the total energy and magnetic moment above the critical temperature

Above in Figure 2, we see that below the critical temperature the magnetic moment quickly reaches a stable equilibrium with all the spins aligned in the same direction. This corresponds to the energetically favourable ferromagnetic configuration. However in Figure 3, above the critical temperature, we see the magnetic moment continues to fluctuate around 0, corresponding to 50% of the spins up and 50% spins down, remaining in a disordered state. This corresponds to the but less entropically favourable paramagnetic state, where there is no net magnetic moment by itself until a magnetic field is applied, at which point the magnetic moment will be proportional to the magnetic field.

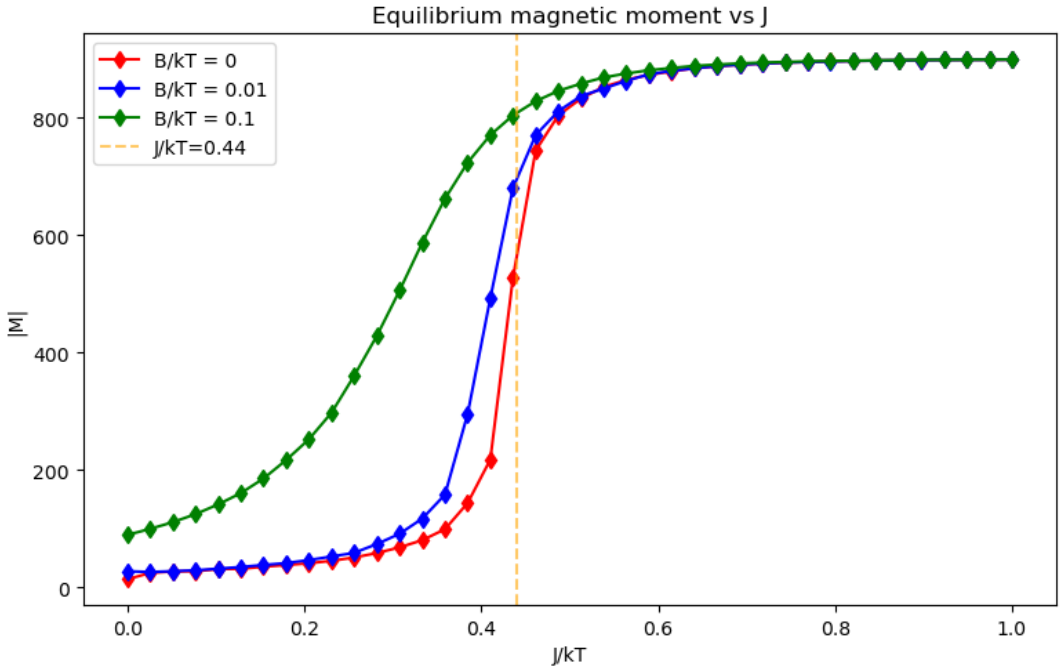

Figure 4: Plot of average magnetic moments at equilibrium for different temperatures

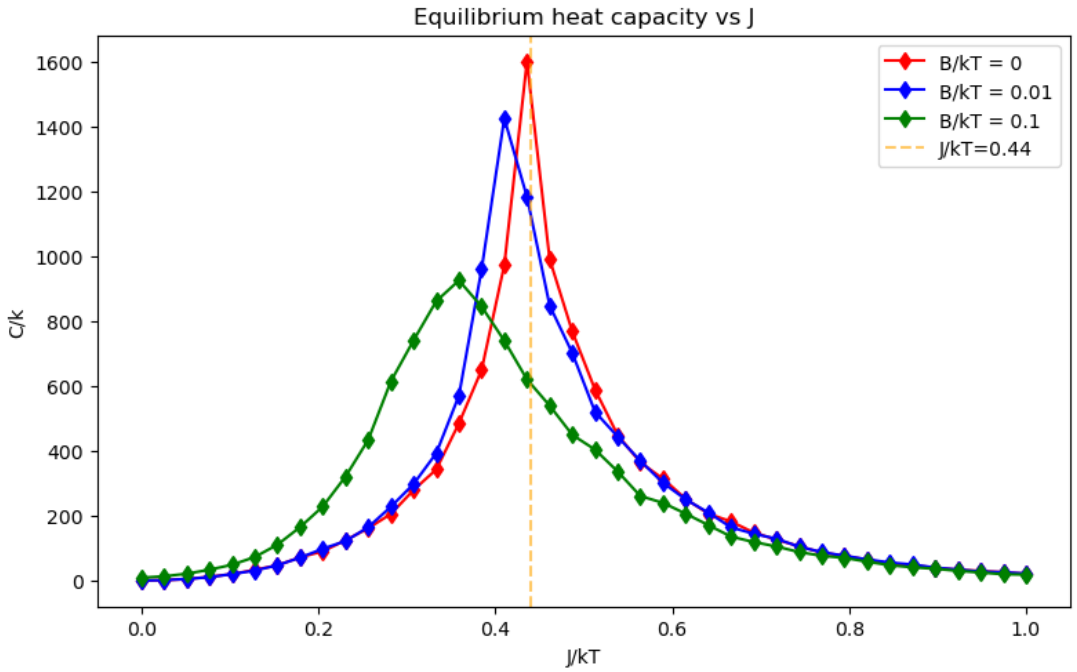

Figure 5: Plot of average heat capacities at equilibrium for different temperatures

Above in Figure 4 and Figure 5, we have run the simulation for 5000 sweep iterations for a 30x30 grid, and averaged the total magnetic moment and heat capacities after 2000 iterations to allow the system to reach equilibrium, then repeated this process for $\frac{J}{k_B T}$ between 0 and 1, for $\frac{B}{k_B T} = 0, 0.01, 0.1$, and plotted the results. We see that for $B = 0$, at $\frac{J}{k_B T} = 0.44$, there is a sharp transition in magnetic moment, and a sharp peak in heat capacity, corresponding to the expected phase transition. For a non-zero applied magnetic field, this is transition is less pronounced and occurs at a higher temperature. We also see that above the critical temperature, the system behaves like a paramagnet, as the magnetic moment increases as the applied magnetic field increases.

5 Conclusion

In this practical we have investigated ferromagnetism by simulating the Ising model using the Metropolis-Hastings algorithm. We have investigated how the model evolves for temperatures below the critical temperature, slowly becoming more ordered until all spins are aligned, while above the critical temperature the model remains disordered. We observed the existence of a phase transition, where the material spontaneously gains a magnetic moment, and there is a spike in heat capacity near the critical temperature. We see that when a magnetic field is applied, this transition becomes less pronounced and occurs at a higher temperature. Although this behaviour is superficially similar to ferromagnetism, it does not explain all aspects of ferromagnetism, for example in real ferromagnets, spins will align to form small domains, however it remains energetically favourable for these domains to remain unaligned until a magnetic field is applied, and so no spontaneous magnetic moment is formed in the absence of an external field. The disadvantages of the Metropolis algorithm have been discussed in Section 3, particularly the burn in time required for convergence, especially near the critical point. Alternative algorithms could be used for this simulation, such as the Wolff algorithm, which will flip clusters of spins instead of individual spins.

References

[1] David J. Griffiths. *Introduction to Electrodynamics*, pages 288–291. Cambridge University Press, 2017.

[2] Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Phys. Rev.*, 65:117–149, Feb 1944.

[3] Stephen J. Blundell and Katherine M. Blundell. *Concepts in Thermal Physics*. Oxford University Press, 10 2009.

[4] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, USA, 3 edition, 2007.

[5] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, USA, 2002.

A Python Code

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation
from matplotlib.colors import ListedColormap
from scipy.signal import convolve2d
import numba
```

[ ]:
```
N = 30
spins = np.random.choice([-1, 1], (N, N))

# Visualise a random arrangement of spins

cmp = ListedColormap(['red', 'blue'])
plt.imshow(spins, cmap=cmp, vmin=-1, vmax=1)
plt.tick_params(bottom=False, top=False, left=False,
                right=False, labelbottom=False, labelleft=False)
plt.show()
```

[ ]:
```
# Performs the Metropolis algorithm, sweep through the lattice and calculate
# acceptance ratio r from energies, and flip spin with probability r
# Use numba (compiles Python to machine code) for almost 100x speedup
# Input: S_init (initial matrix of spins), JkT, BkT (values of J/kT and B/kT)
# Output S_next (matrix of spins after sweep iteration)

@numba.jit(nopython=True)
def sweep(S_init, JkT, BkT):
    S_next = np.copy(S_init)
    for iy, ix in np.ndindex(S_next.shape):
        spinCoupling = S_next[iy, ix] * S_next[(iy + 1) % N, ix]
        spinCoupling += S_next[iy, ix] * S_next[iy - 1, ix]
        spinCoupling += S_next[iy, ix] * S_next[iy, (ix + 1) % N]
        spinCoupling += S_next[iy, ix] * S_next[iy, ix - 1]
        deltaE = 2 * JkT * spinCoupling - 2 * BkT * S_next[iy, ix]
        r = np.exp(-deltaE)
        rand = np.random.uniform()
        if r > rand:
            S_next[iy, ix] *= -1
    return S_next
```

[ ]:
```
neighbours = np.array([[0,1,0],
                       [1,0,1],
                       [0,1,0]])

# Calculate energy by adding up spin-spin interaction and applied magnetic␣
↪field
# Remember to divide first term by 2 to account for double counting of pairs
# Use convolution for efficient neighbour sum
def energy(spins, JkT, BkT):
    convolve = convolve2d(spins, neighbours, mode='same', boundary='wrap')
```

```
    return -JkT * (spins * convolve).sum() / 2 + BkT * spins.sum()
```

[ ]:
```
# Simulate the system for particular values of JkT and BkT
# and store magnetic moments and energies

spinsList = []
magneticMoments = []
energies = []

JkT = 0.5
BkT = 0.0

n_iterations = 2000

spins = np.random.choice([-1, 1], (N, N))

for i in range(n_iterations):
    spinsList.append(spins)
    magneticMoments.append((spins.sum()))
    energies.append(energy(spins, JkT, BkT))
    spins = sweep(spins, JkT, BkT)

energies = np.array(energies)
magneticMoments = np.array(magneticMoments)
```

[ ]:
```
# Plot the magnetic moment and energy

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8,4))

fig.suptitle(f"{N}x{N} grid, $J/k_BT$ = {JkT}, $B/k_BT$ = {BkT}")

ax1.set_title("Energy")
ax1.plot(energies)
ax1.set_xlabel("Sweep iteration")
ax1.set_ylabel("$E/k_BT$")

ax2.set_title("Magnetic Moment")
ax2.plot(magneticMoments)
ax2.set_xlabel("Sweep iteration")
ax2.set_ylabel("$M$")

fig.tight_layout()

plt.show()
```

[ ]:
```
# Visualise the lattice at specific time intervals

times = [0, 1, 5, 10, 25, 50, 100, 200, 500, 1000]

fig, axs = plt.subplots(2, 5, figsize=(12,4))
```

```
for i, ax in enumerate(axs.reshape(-1)):
    ax.set_title(f"t={times[i]}")
    ax.tick_params(bottom=False, top=False, left=False,
                   right=False, labelbottom=False, labelleft=False)
    ax.imshow(spinsList[times[i]], cmap=cmp, vmin=-1, vmax=1)

plt.show()
```

[ ]:
```
fig, [ax1, ax2] = plt.subplots(2, 1, figsize=(8,10))

n_iterations = 5000

BList = [0, 0.01, 0.1]
JList = np.linspace(0, 1, 40)

colors = ["red", "blue", "green"]

# Simulate system for various values of JkT and BkT
for i, BkT in enumerate(BList):
    avgMagMoments = []
    avgHeatCapacities = []
    avgSusceptibilities = []

    for JkT in JList:
        magneticMoments = []
        energies = []
        spins = np.random.choice([-1, 1], (N, N))

        for n in range(n_iterations):
            magneticMoments.append(np.abs(spins.sum()))
            energies.append(energy(spins, JkT, BkT))
            spins = sweep(spins, JkT, BkT)

        # Calculate averages of heat capacity and magnetic moment at␣
↪equilibrium
        # Wait roughly 2000 iterations for burn in period

        magneticMoments = np.array(magneticMoments)
        energies = np.array(energies)
        avgMagMoments.append(np.mean(magneticMoments[2000:]))
        avgHeatCapacities.append(np.var(energies[2000:]))

    # Plot heat capacity/magnetic moment vs JkT for different values of BkT
    ax1.plot(JList, avgMagMoments, c=colors[i],
             marker="d", label=f"$B/k_BT$ = {BkT}")
    ax2.plot(JList, avgHeatCapacities, c=colors[i],
             marker="d", label=f"$B/k_BT$ = {BkT}")

criticalPoint = 0.44

ax1.axvline(criticalPoint, color="orange", alpha=0.6,
```

```
            linestyle='--', label=f"$J/k_BT$={criticalPoint}")
ax1.set_xlabel("$J/k_BT$")
ax1.set_ylabel("$|M|$")
ax1.set_title("Equilibrium magnetic moment vs $J/k_BT$")
ax1.legend()

ax2.axvline(criticalPoint, color="orange", alpha=0.6,
            linestyle='--', label=f"$J/k_BT$={criticalPoint}")
ax2.set_xlabel("$J/k_BT$")
ax2.set_ylabel("$C/k_B$")
ax2.set_title("Equilibrium heat capacity vs $J/k_BT$")
ax2.legend()

fig.tight_layout()

plt.show()
```

[ ]:
```
# Extra code for creating animation of the lattice of spins
fig, ax = plt.subplots()
ax.tick_params(bottom=False, top=False, left=False,
               right=False, labelbottom=False, labelleft=False)

grid = ax.imshow(spins, cmap=cmp, vmin=-1, vmax=1)

def animate(i):
    grid.set_data(spinsList[i])
    return grid

anim = animation.FuncAnimation(fig, animate,
                               frames=n_iterations, interval=50)
anim.save("ising_model.mp4")
```