# Two-Dimensional Ising Model Simulation

PHY2027 – Scientific Programming in C – Final Project Report

Miguel de Sousa

December 12, 2025

## 1 Theory

[1] The Ising model, introduced by Ernst Ising in 1925, is a mathematical model consisting of a lattice of 'spin' variables $\sigma_\alpha$, which can take discrete values of $\pm 1$. These spins represent magnetic dipole moments of atomic spins that can be in one of two states: up (+1) or down (-1). The model is used to study phase transitions and critical phenomena in statistical mechanics. [1] Any two of these spins have a mutual interaction energy:

$$-E_{\alpha\beta}\sigma_\alpha\sigma_\beta \tag{1}$$

[1] Additionally, an external magnetic field $B$ can interact with each spin, contributing an energy term of

$$-B\sigma_\alpha \tag{2}$$

The total energy of a configuration of spins on a lattice can be expressed as

$$E = -\sum_{\langle\alpha,\beta\rangle} E_{\alpha\beta}\sigma_\alpha\sigma_\beta - B\sum_\alpha \sigma_\alpha \tag{3}$$

where the first sum is over all pairs of neighbouring spins. The Ising model exhibits a phase transition at a critical temperature $T_c$, below which the spins tend to align, resulting in spontaneous magnetisation, and above which the spins are disordered.

## 2 Implementation

### 2.1 Data Structure and Initialisation (Part a)

I began by defining a structure type to represent the 2D Ising lattice, which included all necessary parameters as structure members for the model, such as its N-dimensionality, values for temperature $T$, external magnetic field $B$, total energy and magnetisation of the lattice, and a pointer to an array of spins.

Lines 18–20: A char type is assigned to Spin, with constants for SPIN_UP and SPIN_DOWN defined using this new type. The char data type forces each spin to occupy only 1 byte of memory, which is efficient for large $N \times N$ lattices. The Spin type is used throughout the code, so defining it at the start improves readability.

Lines 22–30: The IsingLattice structure type is created with members for lattice size $N$, a pointer to the spin array, temperature $T$, external magnetic field $B$, current step number, total energy, and total magnetisation. Structures are useful for grouping related data together, so this is ideal for representing the lattice and its properties. I used double for $T$, $B$, energy, and magnetisation to allow for fractional values and higher precision in calculations. The step member is an unsigned long long to allow a large number of Monte Carlo steps. The spin pointer defined in the structure is assigned the newly created Spin data type.

After defining the structure type, I implemented the initialisation function to set up the lattice with random $N \times N$ spins.

Lines 145–150: The create_function takes lattice parameters as arguments and allocates memory for the new lattice instance using malloc and the size of the IsingLattice structure type. It checks whether the allocation was successful, printing an error message and exiting if it was not. The structure type pointer is initialised within the function as *lattice.

Lines 152–158: The input parameters are assigned to the corresponding structure members in the IsingLattice using structure pointer notation.

Lines 160–164/169–178: Memory is allocated for the spin array within the lattice structure using malloc. The size is $N \times N$ multiplied by the size of a char (1 byte), since each spin is represented as a char. It checks for successful allocation, freeing the previously allocated lattice memory if the spin allocation fails. I then assigned each lattice spin array value to a random value of SPIN_UP or SPIN_DOWN using a count-controlled loop with boundaries at 0 and $N \times N$. The rand() function was used as the randomiser, with modulus 2 returning either 0 or 1, mapped to either SPIN_UP or SPIN_DOWN. The energy and magnetisation values were also initialised using the energy and magnetisation functions.

Line 166: Finally, the function returns a pointer to the newly created and initialised lattice instance.

## 2.2 Energy and Magnetisation Calculation (Part a)

I proceeded to create a total energy function for my Ising model using the Hamiltonian described in the theory section. The ising_hamiltonian takes the $N$ value of the lattice and initialises a local variable energy with a double type to store all energy components.

Lines 184–186: The code uses nested for loops to iterate through every spin in the lattice, the first iterating through rows and the second through columns. The $s$ value, assigned the Spin datatype, retrieves the current spin values for the lattice as a one-dimensional representation of the two-dimensional array.

Lines 188–194: I iterate through every spin value ($\sigma$) in the lattice and sum the energy contribution from the two possible sources: nearest-neighbour interactions and the external magnetic field ($B$), if present. This task is made efficient by mapping the two-dimensional lattice onto a single one-dimensional array, allowing for faster index lookups and improving efficiency during the summation loop. Each value is added to the double energy variable, and the total energy is returned from the function.

Once the total energy function was created, I began working on the function to output normalised values of magnetisation.

## 2.3 Metropolis Algorithm (Part b)

The Metropolis algorithm is the function describing the evolution of the system. It decides whether a randomly chosen spin is likely to flip based on probability.

Lines 222–224: Using rand(), random $i$ and $j$ locations are chosen to pinpoint a spin value within the lattice in an unbiased manner.

Line 226: Instead of calculating the total energy again, I used the chosen spin location and the change in energy function to compute the energy change if the spin flip were to occur.

Lines 228–241: If the energy change is $\leq 0$, then the energy change is favourable, and the spin flip is always accepted. Nothing is accepted at absolute zero temperature, so that case can be ignored here. The interesting case is when the energy change $\geq 0$. In this situation, the acceptance follows a Metropolis criteria. Using the rand() function again, I created a randomised $r$ value for each iteration of the algorithm, and when $r$ is less than the acceptance probability (Metropolis criteria), the spin flip is successful.

## 2.4 Visualisation Method (Part c)

For my simulation, I decided to take guided user inputs using switch cases to choose standard Ising states. The program begins by giving users flexibility with their choices.

Lines 40–115: The user is prompted with multiple input options for $N$, $T$, and $B$. These values are selected as standard cases used to extract information from the simulation, with temperature being grouped into Low, Critical, and High to view extreme states of the Ising simulation. All inputs are sanitised, ensuring that the user enters only a single valid value. The default case rejects any invalid input. Users also have a custom option. To avoid datatype issues, a temporary integer CASE was created to handle all user selections, since variables such as $T$ with double type cannot be used for switch cases.

Lines 117–143: The program checks for a valid proceed option. If true, a new instance of an Ising lattice is created with the previously selected $N$, $T$, and $B$ values. Once the lattice is initialised, the main simulation begins. I created a loop defining the Monte Carlo steps (simulation time), with the inner loop iterating through every lattice value. Each Monte Carlo step acts as a sweep of the lattice where every spin undergoes a possible flip from the Metropolis algorithm. Each step updates the lattice directly, and snapshots are taken in increments of 200 steps until the loop finishes. All information passed to the Python program is handled by the write_csv function.

Lines 244–274: The write_csv function first checks whether the ising_data.csv file has opened successfully before proceeding. The function takes all values of the Ising lattice, including a nested sweep of the lattice spin values, and writes them to a single CSV file. This includes updated magnetisation, energy, and spin values.

Lines 134–137: Once the simulation CSV is complete, the user is notified and the lattice is freed from memory.

# 3 Program Exploration and Results

## 3.1 Temperature Effect with Zero External Field ($B = 0$)

### 3.1.1 Low Temperature $\approx 1.5Jk_B^{-1}$

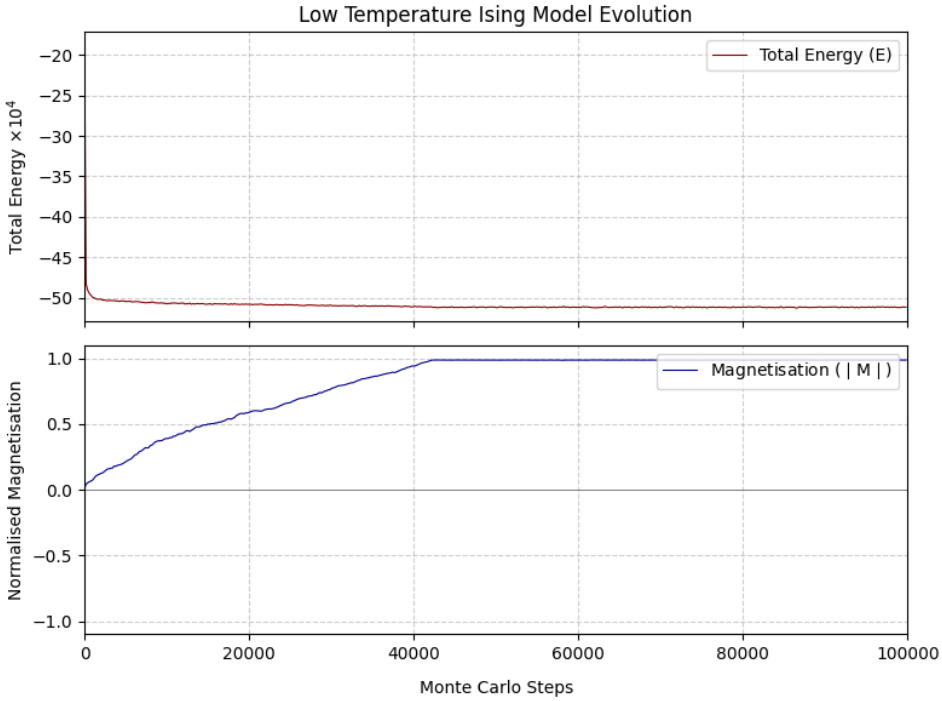


Figure 1: Ising Model Simulation at Low Temperature ($T \approx 1.5Jk_B^{-1}$)

**View Ising Low-Temperature Simulation (Online)**

At a temperature of 1.5, the Ising model simulation is below its critical temperature and therefore lies within the ferromagnetic phase. In this phase, the spins tend to align in the same direction, resulting in a net magnetic moment. As the simulation progresses, we see the formation of large domains of aligned spins, leading to a normalised magnetisation approaching 1. Thermal fluctuations are minimal at this low temperature, so the spins remain mostly stable once they align.

### 3.1.2 Critical Temperature $\approx 2.269 J k_B^{-1}$ [2]

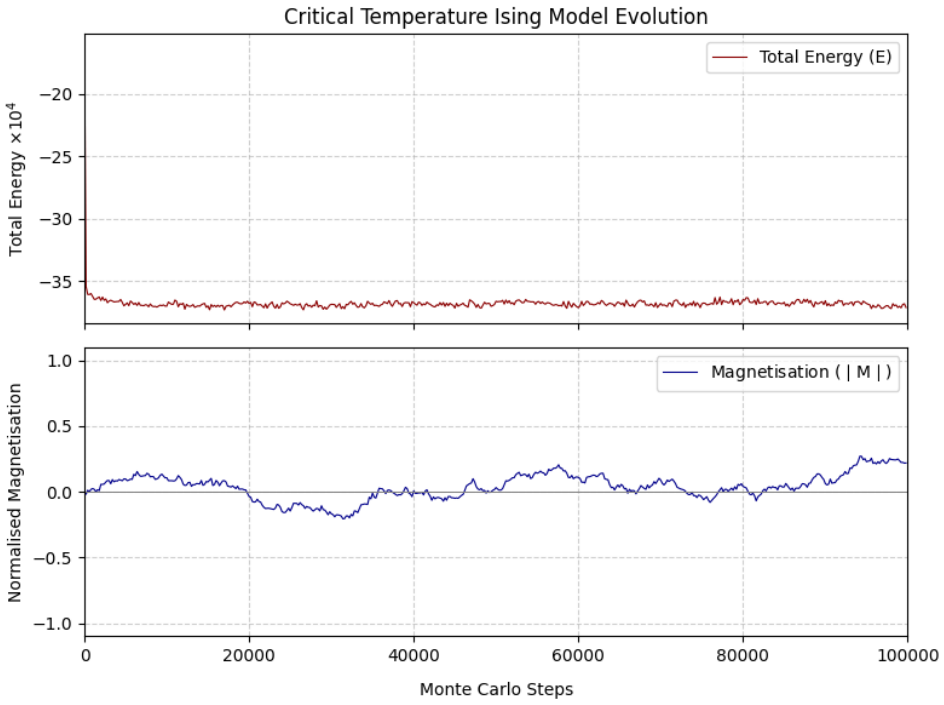


Figure 2: Ising Model Simulation at Critical Temperature ($T \approx 2.269 J k_B^{-1}$)[2]

**View Ising Critical-Temperature Simulation (Online)**

At the critical temperature of approximately 2.269, the Ising model undergoes a phase transition from the ferromagnetic to the paramagnetic phase. In this simulation, many fluctuations in spin alignment occur as the system becomes highly sensitive to thermal energy. Large clusters or domains of aligned spins form and disintegrate frequently, leading to a normalised magnetisation that hovers around zero. This behaviour is characteristic of the critical point and remains consistent even when the simulation is scaled to larger lattice sizes.

### 3.1.3 High Temperature $\approx 7.5Jk_B^{-1}$

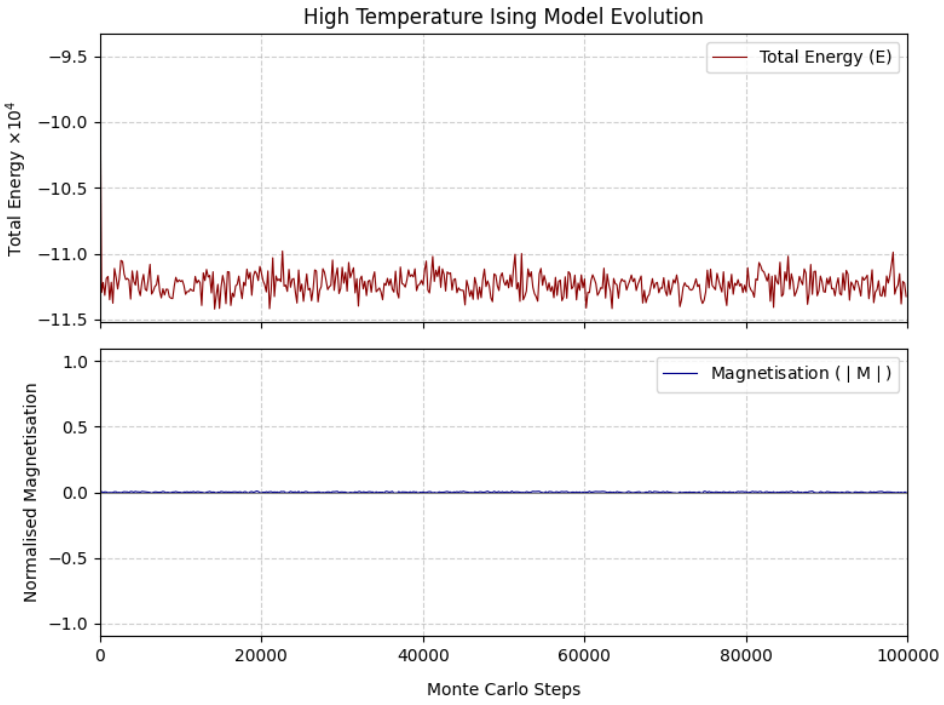


Figure 3: Ising Model Simulation at High Temperature ($T \approx 7.5Jk_B^{-1}$)

**View Ising High-Temperature Simulation (Online)**

At a high temperature of 7.5, the Ising model is well above its critical temperature and remains in the paramagnetic phase. In this phase, thermal fluctuations dominate, causing the spins to become randomly oriented with no possibility of order. As the simulation runs, the spins do not form significant clusters, resulting in a normalised magnetisation that stays close to zero. This high temperature produces a disordered state for the entire simulation.

## 3.2 Effect of External Magnetic Field ($B \neq 0$)

**View Ising Field Simulation (Online)**

When an external magnetic field is applied to the Ising model, it influences the alignment of the spins in the lattice. In this simulation, I observed the effects when a moderate external field ($B = 1.0$) is applied at the critical temperature. The external field biases the spins to align in its direction, leading to a net magnetisation even at the critical temperature, where normalised magnetisation normally hovers around zero. Interestingly, the presence of the external field suppresses the large fluctuations seen in the previous critical simulation; however, the system still shows the creation and destruction of small clusters, just slightly larger than those in the high-temperature simulation without a field. This means that the external magnetic field stabilises the spins at a constant value for the simulation duration, unlike the low-temperature simulation.

## 3.3 Boundary and Neighbour Effects

In the Ising model simulation, I implemented periodic boundary conditions to ensure that the lattice behaves as if it were infinite. This means that spins on one edge of the lattice interact with spins on the opposite edge, effectively "wrapping around" the lattice. Without this boundary condition, the energy calculations would be inaccurate because edge cases would have fewer neighbours. The boundary conditions also ensure that the Ising model accurately represents physical systems; without them, theoretical values (such as the critical temperature $T = 2.269$ [2]) would be shifted slightly downward due to the reduction in total system energy.

# 4 Conclusion

The project successfully implemented a 2D Ising model simulation using the C programming language, with all plotting and analysis handled by Python programs.

I made the code run efficiently in two main ways:

Memory: I used the simple char data type for the spins so that each one only took up 1 byte of memory. This was essential for handling large $N \times N$ lattices, with 262,144 total spins for $N = 512$. If I had used an int (4 bytes), the spin lattice alone would have used 1,048,576 bytes (1 MB) of memory. While this may not seem significant, the Monte Carlo simulation has 100,000 steps, so the file size contribution for the spins alone would have been 500 MB.

Speed: The Metropolis algorithm improved the speed of the simulation. It computes only the small change in energy ($\Delta H$) needed to flip a spin, instead of recalculating the full energy of the entire lattice at every step.

The program exploration showed the expected physical behaviours:

- **Temperature Effect:** At low temperatures ($T \approx 1.5$), the spins aligned quickly, indicating the ferromagnetic phase. At high temperatures ($T \approx 7.5$), the spins were completely random (paramagnetic phase), clearly demonstrating the phase transition.
- **External Field:** Applying a magnetic field caused the whole lattice to align with the field, even at the critical temperature, showing how the field biases the system.

The simulation successfully reproduced the expected behaviours at various temperatures and external magnetic field strengths, demonstrating how the model can be used to study phase transitions and magnetic properties of materials.

## 4.1 Future improvements

**Parallelisation:** Implementing parallel processing (e.g., using threads) could significantly speed up the simulation for larger lattice sizes, since the creation and writing of the large CSV files can be time-consuming.

**Better Visualisation:** Integrating real-time updating plots into the simulation would enhance user experience, allowing users to see the lattice evolution as it happens.

**Python API:** Creating a way for the C program to run the Python file directly would remove the need for the user to manually execute the plotting files after the simulation finishes.

# References

[1] B. M. McCoy and T. T. Wu, *The Two-Dimensional Ising Model*, Harvard University Press, Reprint 2014.

[2] G. R. Puspitasari and W. Gunawan, "Monte Carlo Simulation of 2D Ising Model with Cluster Algorithm," *Journal of Physics: Conference Series*, vol. 2543, no. 1, p. 012006, 2023.