# P3800 Project 3: Monte Carlo Simulation of the 2D Ising Model

You are encouraged to be creative in your coding (but make sure you explain what you did), in your exploration of the Ising model and in the presentation of your results.

For background information, see MacKinnon's notes (pdf on course web page) and almost any book on statistical mechanics or statistical physics.

Your task is to write computer code in Fortran or C (or C++ for those who prefer it) based on the Metropolis Monte Carlo method which simulates thermal averages of the nearest-neighbor Ising model on a square lattice having a spin-dependent energy of the form:

$$E = -J \sum_{i,j}^{NN} S_i S_j$$

where $i, j$ is a sum over nearest neighbors (NN) only and $S_i = \pm 1$. You will use periodic boundary conditions and set $J = 1$ and $k_B = 1$ for convenience. This model can be solved exactly and shows a transition between a paramagnetic state where $M = 0$ and a ferromegnetic state where $M \neq 0$ at a transition temperature $T_c = 2/\ln\left(1+\sqrt{2}\right) \simeq 2.269$. Here, $M(T)$ is the magnetization. Since there is no preferred direction (either up or down), it is expedient to average over the absolute value of the sum of individual spins,

$$M = \frac{1}{N} \langle\, |S| \,\rangle$$

with,

$$S = \sum_{i=1}^{N} S_i$$

where $S_i$ is an individual spin, $N$ is the number of spins and the brackets $\langle \ldots \rangle$ indicate a thermal average over Monte Carlo Steps (MCS) as described in class and in MacKinnon's notes: see his Project - Ising Model description starting on page 43 (we do not recommend the mapping to a 1D problem). In addition to $M(T)$, you should also calculate the heat capacity $C_V(T)$ per spin,

$$C_V = \frac{1}{N} \frac{1}{(k_B T^2)} \left[ \langle E^2 \rangle - \langle E \rangle^2 \right]$$

and magnetic susceptibility $\chi(T)$ per spin,

$$\chi = \frac{1}{N} \frac{1}{(k_B T)} \left[ \langle S^2 \rangle - \langle S \rangle^2 \right].$$

You should get simulation results for these quantities for a range of temperatures around $T_c$, something like $0.5 \leq T \leq 4.0$ in steps of 0.05. Recall from class that for an infinite lattice, $C_V(T)$ and $\chi(T)$ should diverge at $T_c$. It is useful to start at the higher $T$ (using either a ferromagnetic or a random configuration for the initial $S_i$) and cool the system down, using the final configuration from the previous $T$ as the initial configuration of the new $T$. Alternatively, try increasing the temperature instead (start with all spins up in this case). It is also best to discard the first 10% of the runs (MCS0) for thermal equilibration before using the results to get thermal averages. You should consider the effect of different lattice sizes $L \times L$ with $L$ including (at least the) sizes 4, 12, 36 and 100, as well as the impact of using different number of MCS (say, 100, 1000 and 10000). NOTE, it is better to consider one MCS to consist of $L \times L$ single MC spin flips, i.e., a "sweep" of the lattice. Thus, for different lattice size, one MCS will allow, on average, every spin the chance to flip and will serve as a better measure of "time" over which the system runs.

Below is an example of some old (and somewhat terse) Fortran code for the 3D Ising model which contains parts you may find inspiring. It calculates $M$ at one particular value of $T$. Your code can be structured in a clearer manner. Consider the following when coding.

- Have a separate part that stores the five possible values of $\Delta E$ (see MacKinnon).
- Calculate the energy for a given configuration of spins using a subroutine.
- Choose spins $S_i$ to flip (or not) randomly instead of systematically.
- Flip a spin or not based on the `if` statements that avoid checking against a random number if it is not really necessary.

In your write-up you should discuss any particular coding issues you think are relevant. There should be lots of graphs of the quantities vs T for different lattices sizes (consider putting data for the same quantity for different $L$ on the same graph) and different MCS. Maybe even a plot of your estimates for $T_c$ vs $L$ would be useful (try various log-log or semi-log plots). Does it extrapolate to the correct value? How do divergences scale with $N$? Also consider including illustrative results for $S_i$ at each lattice point for the case of $T \ll T_C$ as well as $T \gg T_C$ which should show nearly all up-spins in the former case and nearly equal numbers of up-spins and down-spins in the latter case. Consider using `pyplot.matshow` to create plots of the configurations. Submit your completed project electronically (as with previous projects) and also a hard copy of your write-up (pdf).

For your reference, the website includes reference data for the $M(T)$ and $\chi(T)$ for large systems.

Application of Monte Carlo Method in Stat. Phys. ed. K. Binder (1987)

Table 1.1. Example of a program for a 37*37*37 Ising model of a simple cubic Ising lattice at $T/T_C = 1.4$, calculated for 25 Monte Carlo steps per spin. *Ranset* initializes the random number generator RANF. The energy change is calculated as in (1.5), the flipping probability is taken as $\exp(-\Delta E/k_BT)$. A CDC Cyber 76 took about 3.75 s for this program

```
     DIMENSION IS(37,37,37),EX(13)
     DATA IS/50653*1/
     T=1.40/0.221655
     L=37
     CALL RANSET(1)
     M=L*L*L
     DØ 3 I=1,13,2
   3 EX(I)=EXP(-2*(I-7.)/T)
     DØ 2  ITIME=1,25
     DØ 1 K1=1,L
     K1P1=K1+1
     K1M1=K1-1
     IF(K1.EQ.1)K1M1=L
     IF(K1.EQ.L)K1P1=1
     DØ 1 K2=1,L
     K2P1=K2+1
     K2M1=K2-1
     IF(K2.EQ.1)K2M1=L
     IF(K2.EQ.L)K2P1=1
     DØ 1 K3=1,L
     K3P1=K3+1
     K3M1=K3-1
     IF(K3.EQ.1)K3M1=L
     IF(K3.EQ.L)K3P1=1
     IEN=7+IS(K3,K2,K1)*(IS(K3M1,K2,K1)+IS(K3P1,K2,K1)+IS(K3,K2M1,K1)
    1                +IS(K3,K2P1,K1)+IS(K3,K2,K1M1)+IS(K3,K2,K1P1))
     IF(EX(IEN).LT.RANF(I)) GØTØ 1
     IS(K3,K2,K1)=-IS(K3,K2,K1)
     M=M+2*IS(K3,K2,K1)
   1 CONTINUE
   2 WRITE (6,4)M,ITIME
   4 FORMAT(2I9)
     STOP
     END
```

$e^{-\Delta E/T}$

Notes:

In 3D, a spin has 6 niehgbours, and there are only 7 possible changes in energy depending on whether there are 0, 1, 2, … or 6 neighbours with the same spin. The related probabilities are stored in an array as a "look-up" table so that exp need not be calculated each time.

This algorithm sweeps through all lattice sites sequentially. Although this does not break detailed balance, your implementation should randomly pick spins to avoid possible kinetic correlations.

If the spin flip is rejected, M remains unchanged.

The algorithm only accumulates to the running sum once N=L*L*L spin flips have been attempted, i.e., after every "sweep" of the lattice.

[illegible] d2

[illegible] par 2

# Fun with ChatGPT

In an attempt to simplify things, I asked ChatGPT to produce some code for it. Here is what I asked, but I'll only show the final outputs.

- Please produce code in C for the 2D Ising model.
- keep going
- The two code fragments you produced were not complete. Could you please produce a complete code for the 2D Ising model.
- I liked the style of the first version of the code. Could you complete the following code? [pasted code]
- Could you please produce a Fortran 90 version of the code?
- The Fortran version has at least one error arising because C is case sensitive and Fortran is not. Could you fix the Fortran version?
- You're using j and J to represent two different variables, but in Fortran j and J are the same.
- Why do you stop generating the reponse without completing the code? Please produce a full version
- still there? [I had closed my laptop.]
- Ok, let's try again. Please produce code in Fortran 90 for the 2D Ising model. Use a random seed based on the time.
- Please don't stop generating code until the equilibration stage and the measuring stage are both finished.
- and use the variabe Jexch for the exchange constant
- keep going!
- . . . (I tried a few other things but did not get much further.)

Below are equivalent C and Fortran versions from ChatGPT for the 2D Ising model. Note that in the Fortran code, the exchange constant is J, which conflicts with integer variable j, and so should be renamed.

### Here is the C code.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define N 32 // lattice size
#define J 1.0 // exchange interaction strength
#define kB 1.0 // Boltzmann constant
#define warmupSteps 1000 // number of steps for equilibration
#define measureSteps 100 // number of steps for measuring observables
#define nTemps 100 // number of temperatures to simulate
#define Tmin 1.0 // minimum temperature
#define Tmax 5.0 // maximum temperature
#define dT (Tmax-Tmin)/(nTemps-1) // temperature step size

int main()
{
    // Initialize the random number generator
    srand(time(NULL));

    // Allocate memory for the spin lattice
    int **lattice;
    lattice = (int **)malloc(N * sizeof(int *));
    for (int i = 0; i < N; i++) {
        lattice[i] = (int *)malloc(N * sizeof(int));
    }

    // Initialize the spin lattice to a random configuration
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (rand() < RAND_MAX/2) {
                lattice[i][j] = 1;
            } else {
                lattice[i][j] = -1;
            }
        }
    }
```

```
// Initialize the observables
double energy = 0.0;
double magnetization = 0.0;

// Loop over temperatures
for (int t = 0; t < nTemps; t++) {
    double T = Tmin + t*dT;
    printf("Temperature = %f\n", T);

    // Equilibration
    for (int i = 0; i < warmupSteps; i++) {
        // Choose a random spin
        int x = rand() % N;
        int y = rand() % N;

        // Compute the energy change if the spin is flipped
        int deltaE = 2*J*lattice[x][y]*
                     (lattice[(x-1+N)%N][y] +
                      lattice[(x+1)%N][y] +
                      lattice[x][(y-1+N)%N] +
                      lattice[x][(y+1)%N]);

        // Decide whether to flip the spin
        if (deltaE < 0 || exp(-deltaE/(kB*T)) > ((double)rand())/RAND_MAX) {
            lattice[x][y] = -lattice[x][y];
            energy += deltaE;
            magnetization += 2*lattice[x][y];
        }
    }

    // Measuring observables
    double sumEnergy = 0.0;
    double sumEnergy2 = 0.0;
    double sumMagnetization = 0.0;
    double sumMagnetization2 = 0.0;
    double sumAbsMagnetization = 0.0;
    for (int i = 0; i < measureSteps; i++) {
        // Choose a random spin
        int x = rand() % N;
```

```
            int y = rand() % N;

            // Compute the energy change if the spin is flipped
            int deltaE = 2*J*lattice[x][y]*
                         (lattice[(x-1+N)%N][y] +
                          lattice[(x+1)%N][y] +
                          lattice[x][(
```

## And here is the Fortran code.

```
program ising_2d

implicit none

integer, parameter :: N=32 ! lattice size
real, parameter :: J=1.0 ! exchange interaction strength
real, parameter :: kB=1.0 ! Boltzmann constant
integer, parameter :: warmupSteps=1000 ! number of steps for equilibration
integer, parameter :: measureSteps=100 ! number of steps for measuring observables
integer, parameter :: nTemps=100 ! number of temperatures to simulate
real, parameter :: Tmin=1.0 ! minimum temperature
real, parameter :: Tmax=5.0 ! maximum temperature
real, parameter :: dT=(Tmax-Tmin)/(nTemps-1) ! temperature step size

integer :: i, j, t, x, y
integer, dimension(N,N) :: lattice
real :: T, energy, magnetization, deltaE
real :: sumEnergy, sumEnergy2, sumMagnetization, sumMagnetization2, sumAbsMagnetization
integer :: acceptCount

! Initialize the random number generator
call srand(time())

! Initialize the observables to zero
energy = 0.0
magnetization = 0.0

! Loop over temperatures
```

```
do t=0, nTemps-1
    T = Tmin + t*dT
    write(*,*) 'Temperature = ', T

    ! Equilibration
    do i=1, warmupSteps
        ! Choose a random spin
        x = mod(rand(), N)
        y = mod(rand(), N)

        ! Compute the energy change if the spin is flipped
        deltaE = 2.0*J*lattice(x,y)* &
            (lattice(mod(x-1+N,N),y) + &
             lattice(mod(x+1,N),y) + &
             lattice(x,mod(y-1+N,N)) + &
             lattice(x,mod(y+1,N)))

        ! Decide whether to flip the spin
        if (deltaE < 0.0 .or. exp(-deltaE/(kB*T)) > rand()/real(RAND_MAX)) then
            lattice(x,y) = -lattice(x,y)
            energy = energy + deltaE
            magnetization = magnetization + 2.0*lattice(x,y)
        endif
    end do

    ! Measuring observables
    sumEnergy = 0.0
    sumEnergy2 = 0.0
    sumMagnetization = 0.0
    sumMagnetization2 = 0.0
    sumAbsMagnetization = 0.0
    acceptCount = 0

    do i=1, measureSteps
        ! Choose a random spin
        x = mod(rand(), N)
        y = mod(rand(), N)

        ! Compute the energy change if the spin is flipped
        deltaE = 2.0*J*lattice(x,y)* &
            (lattice(mod(x-1+N,N),y) + &
             lattice(mod(x+1,N),y) + &
             lattice(x,mod(y-1+N,N)) + &
             lattice(x,mod(y+1,N)))

        ! Decide whether to flip the spin
        if (deltaE < 0.0 .or. exp(-deltaE/(kB*T)) > rand()/real(RAND_MAX)) then
            lattice(x,y) = -lattice(x,y)
            energy = energy + deltaE
            magnetization = magnetization + 2.0*lattice(x,y
```