

Two-Dimensional Ising Model of Ferromagnetism

Magnetism in matter is caused by charged particles moving in closed orbits or spinning around their axes. Recall that a current loop creates a magnetic field according to Ampère's law. A spinning charged particle has a magnetic moment associated with it.

A simple classical approximation to an atomic or electronic magnetic moment is provided by an Ising spin which can take two values

$$s_i = \begin{cases} +1, & \text{represents "spin up"} \\ -1, & \text{represents "spin down"} \end{cases}$$

A two-dimensional magnet can be modeled by a set of N_s spins located on a fixed two-dimensional lattice of sites. For example, we can have a square lattice with L spins in the x direction and L in the y direction such that $L^2 = N_s$.

The force between two magnets falls off rather rapidly, like r^{-3} . So a reasonable approximation is to assume that any spin interacts only with its 4 nearest neighbors—north, south, east and west. The interaction energy can be approximated by

$$E = -J \sum_{\langle ij \rangle} s_i s_j - H \sum_i s_i .$$

If the interaction strength $J > 0$ the system is ferromagnetic: the energy is minimized if the spin point in the same direction $s_i s_j = +1$. If $J < 0$ the system is antiferromagnetic. H represents an external magnetic field which couples to the *magnetization*

$$M = \sum_i s_i ,$$

of the system. The spins prefer to line up with the magnetic field.

Ferromagnetism, Paramagnetism, and Curie Temperature

If $H = 0$, the system can exist in two different phases depending on the temperature T . At low temperatures, the system is permanently magnetized. At sufficiently high temperatures, the magnetization of the system is zero.

There is a critical value of the temperature T_c called the *Curie temperature* at which a *phase transition* between the ferromagnetic (permanently magnetized) and paramagnetic phases occurs.

Monte Carlo Simulation

Monte Carlo simulations involve generating a subset of *configurations* or *samples*, chosen using a random algorithm from a *configuration space*, according to a *probability distribution* or *weight function*. Observables are then computed as *averages* over the samples.

- **Configuration or Sample:** One sample or configuration of the magnet is a particular assignment of spin values, say

$$s_1 = +1, s_2 = -1, s_3 = +1, \dots, s_{N_s} = +1 ,$$

in which each spin is set “up” or “down”. According to statistical mechanics, the average value of an observable is got by weighting each configuration with the *Boltzmann factor*. For example, the average magnetization at some fixed temperature T is given by

$$\langle M \rangle = \frac{\sum_{\text{configs}} M e^{-E/k_B T}}{\sum_{\text{configs}} e^{-E/k_B T}} .$$

- **Configuration Space:** The total number of configurations of the system is enormous even for small numbers of spins. For example if $L = 20$, $N_s = 20^2 = 400$, and

$$\text{No. of configs} = 2^{N_s} = 2^{400} = 2.58 \times 10^{120} .$$

If we tried to enumerate the configurations at a billion per second on a very fast computer, it would take 2.58×10^{111} seconds $= 8.8 \times 10^{103}$ years to compute the average magnetization exactly!

- **Probability Distribution or Weight Function:** The basic idea of a Monte Carlo calculation is to generate a reasonable number of configurations at random. The Boltzmann factor is an exponential function of energy

which can vary enormously. The random configurations are therefore generated with probability determined by this exponential factor:

$$p(s_1, s_2, \dots, s_{N_s}) = \frac{e^{-E(s_1, s_2, \dots, s_{N_s})/k_B T}}{\sum_{\text{configs}} e^{-E/k_B T}} .$$

- **Monte Carlo Average:** The problem now is to generate N statistically independent configurations that are distributed according to the Boltzmann factor:

$$(s_1^{(i)}, s_2^{(i)}, \dots, s_{N_s}^{(i)}), \quad i = 1, 2, \dots, N ,$$

the average magnetization and energy are given by

$$\langle M \rangle = \frac{1}{N} \sum_{i=1}^N M(s_1^{(i)}, s_2^{(i)}, \dots, s_{N_s}^{(i)}) ,$$

$$\langle E \rangle = \frac{1}{N} \sum_{i=1}^N E(s_1^{(i)}, s_2^{(i)}, \dots, s_{N_s}^{(i)}) .$$

Algorithm of Metropolis *et al.*

How does one generate configurations distributed according to the Boltzmann factor? A very ingenious algorithm to do this was discovered by N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, *J. Chem. Phys.* **21**, 1087 (1953). Applied to this model, the algorithm generates the next configuration in the sequence as follows:

- Given a configuration, choose a spin s_i , let $s_{i,\text{trial}} = -s_i$.
 - Compute the change in energy of the system

$$\Delta E = E(s_1, s_2, \dots, s_{i,\text{trial}}, \dots, s_{N_s}) - E(s_1, s_2, \dots, s_i, \dots, s_{N_s}) .$$

- If

$$w = e^{-\Delta E/k_B T} > r ,$$

where r is a uniform random deviate let $s_i \leftarrow s_{i,\text{trial}}$, flip this spin.

- Repeat the above for all N_s spins in the configuration.

Starting from some initial configuration, a suitable number of Monte Carlo steps are taken to allow the system to thermalize. After this the production phase begins and statistical averages are measured.

Monte Carlo Program to Simulate the 2-D Ising Model

```
// Ising Model in two dimensions

#include <cmath>
#include <cstdlib>
#include <iostream>
#include <fstream>
#include "rng.h"

using namespace std;

double J = +1;           // ferromagnetic coupling
int Lx, Ly;              // number of spins in x and y
int N;                   // number of spins
int **s;                 // the spins
double T;                // temperature
double H;                // magnetic field
```

Efficient evaluation of Boltzmann factors

Computing the exponential factor $w = e^{-\Delta E/k_B T}$ at each Metropolis step is expensive. However, it is not hard to see that for a simulation at fixed T and H there are only 10 distinct values for W . Pre-computing these 10 values at the start of the simulation saves the expense of computing the exponential thousands or millions of times during the simulation!

First consider the sum of the 4 neighboring spins:

$$\sum_{\text{neighbors } j} s_j = s_{(i_x+1, i_y)} + s_{(i_x-1, i_y)} + s_{(i_x, i_y-1)} + s_{(i_x, i_y+1)} .$$

This sum takes the following possible values:

1. $\sum_{\text{neighbors } j} s_j = +4$ if all four neighbors point up,
2. $\sum_{\text{neighbors } j} s_j = +2$ if three neighbors point up and one down,
3. $\sum_{\text{neighbors } j} s_j = 0$ if two neighbors point up and two down,
4. $\sum_{\text{neighbors } j} s_j = -2$ if one neighbor points up and three down,
5. $\sum_{\text{neighbors } j} s_j = -4$ if all four neighbors point down.

Since $s_i = \pm 1$, the product takes the same set of five values:

$$s_i \sum_{\text{neighbors } j} s_j = +4, +2, 0, -2, -4 .$$

If the magnetic field $H \neq 0$, the term Hs_i which takes two values $\pm H$ needs to be taken into account. Thus there are 10 different values for w .

```
double w[17][3];           // Boltzmann factors

void computeBoltzmannFactors ( ) {
```

```

for (int i = -8; i <= 8; i += 4) {
    w[i + 8][0] = exp( - (i * J + 2 * H) / T);
    w[i + 8][2] = exp( - (i * J - 2 * H) / T);
}

```

Since array indices in C++ must be ≥ 0 , the first index of the array is computed as

$$8 + 2s_i \quad \sum_{\text{neighbors } j} s_j = 0, 4, 8, 12, 16 ,$$

and the second index of the array is computed as

$$1 + s_i = 0, 2 .$$

```

int steps = 0;                                // steps so far

void initialize ( ) {
    s = new int* [Lx];
    for (int i = 0; i < Lx; i++)
        s[i] = new int [Ly];
    for (int i = 0; i < Lx; i++)
        for (int j = 0; j < Ly; j++)
            s[i][j] = qadran() < 0.5 ?  +1 :  -1;    // hot start
    computeBoltzmannFactors();
    steps = 0;
}

```

Taking one Metropolis step

One of the N spins is chosen at random and the Metropolis step is applied to it. If the spin is at one of the 4 edges of the lattice, then we use periodic boundary conditions to determine its neighbors.

```
bool MetropolisStep ( ) {  
  
    // choose a random spin  
    int i = int(Lx*qadran());  
    int j = int(Ly*qadran());  
  
    // find its neighbors using periodic boundary conditions  
    int iPrev = i == 0 ? Lx-1 : i-1;  
    int iNext = i == Lx-1 ? 0 : i+1;  
    int jPrev = j == 0 ? Ly-1 : j-1;  
    int jNext = j == Ly-1 ? 0 : j+1;  
  
    // find sum of neighbors  
    int sumNeighbors = s[iPrev][j] + s[iNext][j] + s[i][jPrev] + s[i][jNext];  
    int delta_ss = 2*s[i][j]*sumNeighbors;  
  
    // ratio of Boltzmann factors  
    double ratio = w[delta_ss+8][1+s[i][j]];  
    if (qadran() < ratio) {  
        s[i][j] = -s[i][j];  
        return true;  
    } else return false;  
}
```

One Monte-Carlo step per spin

A single Metropolis step will yield a configuration which is highly correlated with the current configuration. In Monte Carlo simulations, it is conventional to take *at least* N Metropolis steps to generate the next configuration. This gives each spin the opportunity to change its state.

```
double acceptanceRatio;

void oneMonteCarloStepPerSpin ( ) {
    int accepts = 0;
    for (int i = 0; i < N; i++)
        if (MetropolisStep())
            ++accepts;
    acceptanceRatio = accepts/double(N);
    ++steps;
}
```

Measuring observables

The following function computes the average magnetization per spin of the lattice:

```
double magnetizationPerSpin ( ) {
    int sSum = 0;
    for (int i = 0; i < Lx; i++)
        for (int j = 0; j < Ly; j++) {
            sSum += s[i][j];
        }
    return sSum / double(N);
}
```

```
}
```

and the energy per spin, taking into account periodic boundary conditions, is computed by the following function:

```
double energyPerSpin ( ) {
    int sSum = 0, ssSum = 0;
    for (int i = 0; i < Lx; i++)
    for (int j = 0; j < Ly; j++) {
        sSum += s[i][j];
        int iNext = i == Lx-1 ? 0 : i+1;
        int jNext = j == Ly-1 ? 0 : j+1;
        ssSum += s[i][j]*(s[iNext][j] + s[i][jNext]);
    }
    return -(J*ssSum + H*sSum)/N;
}
```

Steering the calculation: the main function

The main function starts by getting input from the user.

```
int main (int argc, char *argv[]) {

    cout << " Two-dimensional Ising Model - Metropolis simulation\n"
         << " -----\n"
         << " Enter number of spins L in each direction: ";
    cin >> Lx;
```

```
Ly = Lx;
N = Lx*Ly;
cout << " Enter temperature T: ";
cin >> T;
cout << " Enter magnetic field H: ";
cin >> H;
cout << " Enter number of Monte Carlo steps: ";
int MCSteps;
cin >> MCSteps;
initialize();
```

We first perform thermalization steps to let the system come to thermal equilibrium at the specified temperature.

```
int thermSteps = int(0.2 * MCSteps);
cout << " Performing " << thermSteps
    << " steps to thermalize the system ..." << flush;
for (int s = 0; s < thermSteps; s++)
    oneMonteCarloStepPerSpin();
```

Next we perform production steps, take data, and accumulate averages.

```
cout << " Done\n Performing production steps ..." << flush;
double mAv = 0, m2Av = 0, eAv = 0, e2Av = 0;
ofstream file("ising.data");
for (int s = 0; s < MCSteps; s++) {
    oneMonteCarloStepPerSpin();
    double m = magnetizationPerSpin();
```

```

        double e = energyPerSpin();
        mAv += m; m2Av += m * m;
        eAv += e; e2Av += e * e;
        file << m << '\t' << e << '\n';
    }
    file.close();
    mAv /= MCSteps; m2Av /= MCSteps;
    eAv /= MCSteps; e2Av /= MCSteps;
    cout << " \n Magnetization and energy per spin written in file "
         << " \"ising.data\" " << endl;
    cout << " <m> = " << mAv << " +/- " << sqrt(m2Av - mAv*mAv) << endl;
    cout << " <e> = " << eAv << " +/- " << sqrt(e2Av - eAv*eAv) << endl;
}

```

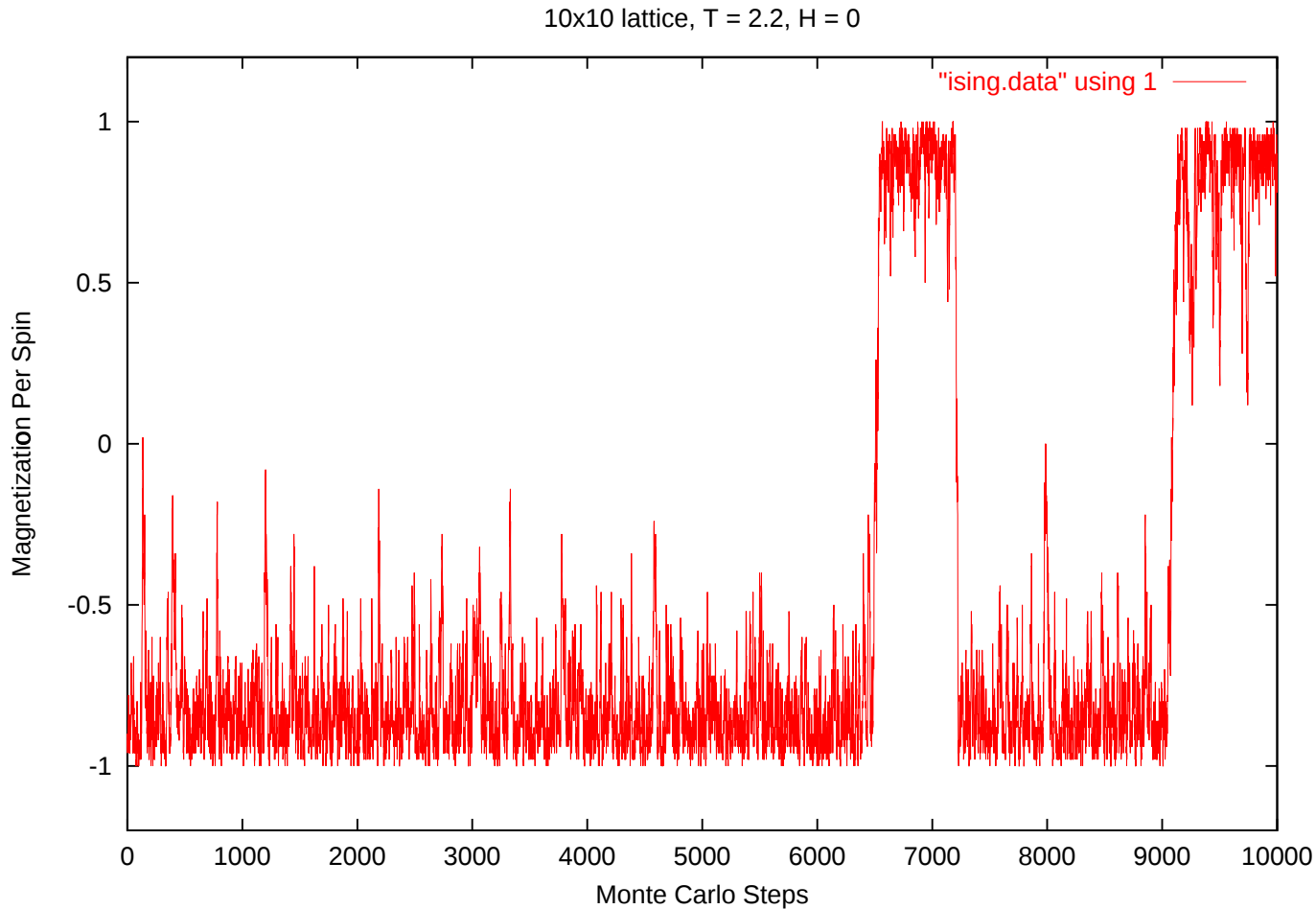
Ferromagnetism below the Curie temperature

In 1941, H.A. Kramers and G.H. Wannier, *Phys. Rev.* **60**, 252 (1941), used a *duality argument* to compute the exact value of the Curie temperature for the two dimensional Ising model on a square lattice. They showed that

$$\frac{k_B T_c}{J} = \frac{2}{\log(1 + \sqrt{2})} = 2.269 \dots$$

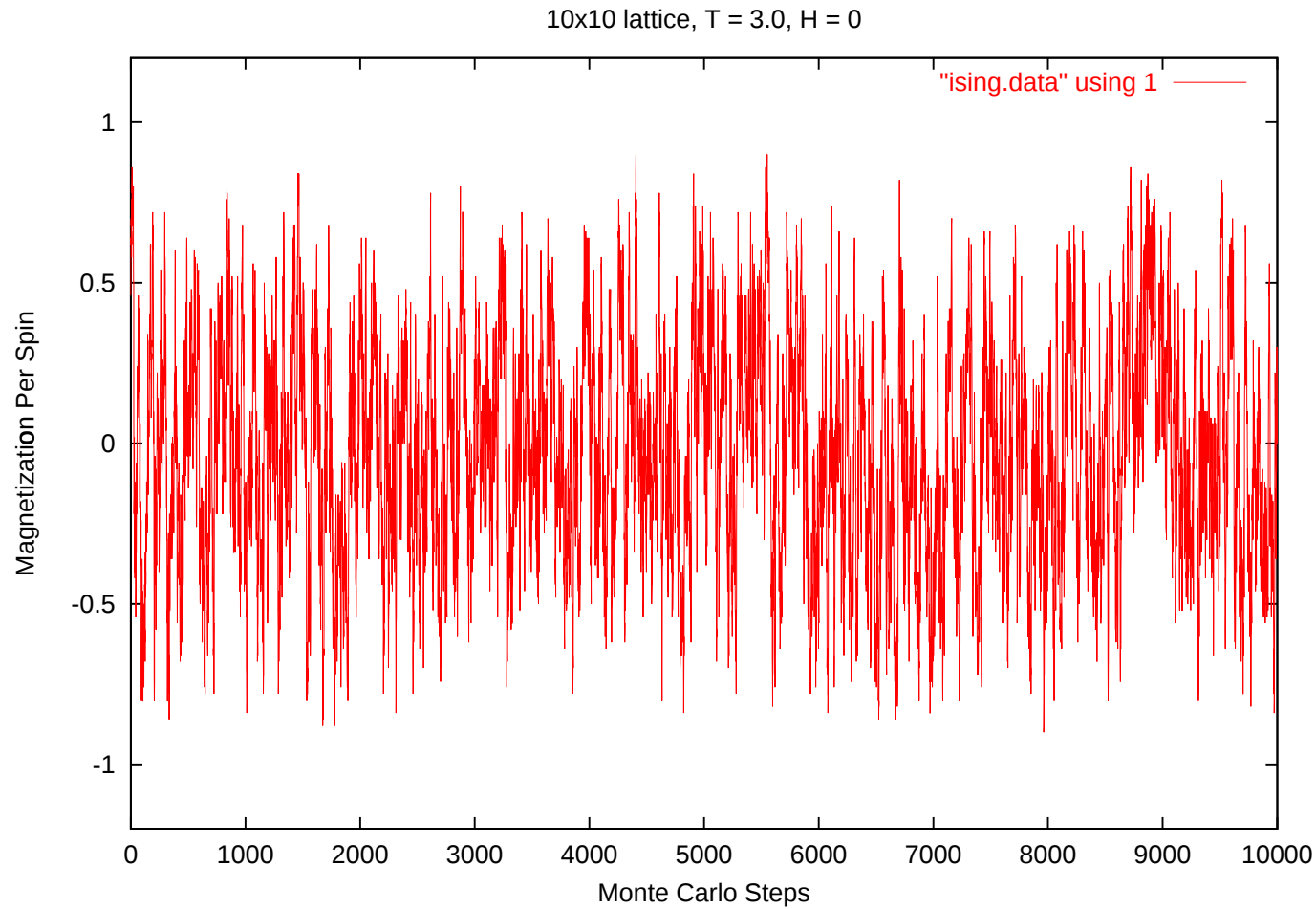
In 1944, Lars Onsager, *Phys. Rev.* **65**, 117 (1944), solved the model exactly in the thermodynamic limit $N \rightarrow \infty$ and zero magnetic field $H = 0$. He showed, for example that the magnetization per spin

$$m = \lim_{N \rightarrow \infty} \frac{\langle \sum_i s_i \rangle}{N} = \begin{cases} \left[1 - \left\{ \sinh \left(\frac{2J}{k_B T} \right) \right\}^{-4} \right]^{1/8}, & \text{for } T \leq T_c \\ 0, & \text{for } T > T_c \end{cases}$$



The data show the magnetization per spin at $T = 2.2$, which is below the Curie temperature as a function of Monte Carlo step. The system appears to be a permanent magnet with negative magnetization. However, sudden reversals of the magnetization occur from time to time. It can be shown that a *finite size* system cannot have a permanent non-zero magnetization in zero magnetic field. Note also the fluctuations in the magnetization, which are a measure of the *magnetic susceptibility* of the system.

The next plot shows the magnetization per spin at $T = 3.0$ which is above the Curie temperature. As expected,



the average magnetization is zero, which shows that the system is a *paramagnet*.