Tallinn University of Technology

Department of Physics

Indrek Mandre
(YAFM 081797)

THE ISING MODEL

course project in

Simulation of Physical Processes

Tallinn 2008

Substance	Formula	Force
Diamagnetic		
Water	H_2O	-22
Copper	Cu	-2.6
Diamond	C	-16
Graphite	C	-110
Paramagnetic		
Sodium	Na	20
Nickel sulfate	$NiSO_4$	830
Liquid oxygen	O_2	7500 (90K)
Ferromagnetic		
Iron	Fe	400000
Magnetite	Fe_3O_4	120000

Table 1: Relative forces on substances [Purcell].

1 Magnetic Fields in Matter

When a substance is placed into a nonuniform magnetic field, a force can be measured acting on the matter. As can be seen in table 1, for different substances the force can vary in direction and magnitude. A change in the net alignment of magnetic dipoles within the matter happens, and the medium becomes magnetically polarized, or magnetized. This effect is divided into three types:

1. *Diamagnetism* - the magnetization happens opposite to the applied magnetic field **B** and the substance is repulsed. Diamagnetism is caused by the change in the orbital magnetic dipoles due to the electron speeding up or slowing down in an external magnetic field.

2. *Paramagnetism* - the dipoles within the matter align themselves parallel to the field **B** and the substance is attracted towards the external magnet. Paramagnetism is caused by the slight change of the orbital planes of the electrons and by the polarization of electron spins.

3. *Ferromagnetism* - the magnetization is parallel to the **B**, very strong and the substance will retain their magnetization even after the external field has been removed. For these the magnetization is not determined by the present field but by the whole magnetic history of the object. The most common ferromagnetic materials are iron, nickel and cobalt. Ferromagnetism is caused by the spontaneous alignment of electron spins.

2 Paramagnetism and Diamagnetism[1]

An external uniform magnetic field applied on a loop of current will generate torque

$$\boldsymbol{\tau} = \mathbf{m} \times \mathbf{B}$$

where $\mathbf{m} = IS\hat{\mathbf{n}}$ is the magnetic dipole moment of the loop of current. This torque will try to align the dipole vector along the magnetic field. Because the magnetic field is uniform the net force on the loop will be 0:

$$\mathbf{F} = I \oint (d\mathbf{l} \times \mathbf{B}) = I \left(\oint d\mathbf{l} \right) \times \mathbf{B} = 0.$$

In case of a nonuniform magnetic field there may be a net force on the current loop. For an infinitesimal loop, with dipole moment $\mathbf{m}$, in a field $\mathbf{B}$, the force is

$$\mathbf{F} = \nabla (\mathbf{m} \cdot \mathbf{B}).$$

When taking the Bohr model of an atom we can see that electrons revolve around the nucleus. This circular movement can be looked at as a small current loop and approximates as steady current. If the electron moves at speed v at radius R, then the period of movement is $T = 2\pi R/v$ and the current is

$$I = \frac{e}{T} = \frac{ev}{2\pi R}.$$

From this we get the orbital dipole moment of an atom:

$$\mathbf{m} = -\frac{1}{2} evR\hat{\mathbf{n}}.$$

Like any other magnetic dipole, this one is subject to a torque $(\mathbf{m} \times \mathbf{B})$ when the atom is placed in a magnetic field. The orbits of electrons don't tend to tilt very much though and the contribution from this to the paramagnetism is small.

There is another more significant effect on the orbital motion: the electron speeds up or slows down in an external magnetic field. This is due to the Lorenz force $-e(\mathbf{v} \times \mathbf{B})$. Assuming the magnetic field is perpendicular to the plane of the orbit, the change in speed is

$$\Delta v = \frac{eRB}{2M}.$$

This change in speed happens when the magnetic field is turned on or off. A change in orbital speed means a change in the magnetic dipole moment:

$$\Delta m = -\frac{e^2 R^2}{4M} B.$$

The change is in the negative direction of $\mathbf{B}$. Now ordinarily all the electrons in the matter are randomly oriented, and the orbital dipole moments cancel out. But

[1] Based on [Griffiths].

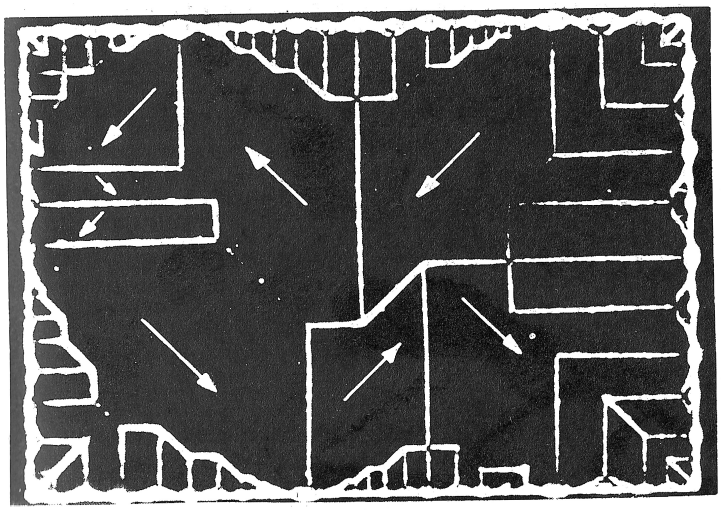

Figure 1: Ferromagnetic domains [Griffiths].

in the presence of a magnetic field, each atom pick up a little "extra" dipole moment, and these increments are all anti-parallel to the field. This will result in a net magnetic field generated by the magnetic dipoles that opposes the applied magnetic field. This is the mechanism responsible for diamagnetism. It is a universal phenomenon and affects all atoms. However, it is typically much weaker than paramagnetism and is observed mainly in atoms with even numbers of electrons where paramagnetism is usually absent.

3 Ferromagnetism[2]

In ferromagnetism, the magnetic field generated by matter is caused by the spins of unpaired electrons. Each of those spins "likes" to point in the same direction as its neighbors. The reason for that is essentially quantum mechanical. The correlation is so strong that it virtually aligns 100% of the unpaired electron spins.

The alignment usually occurs in relatively small patches, called domains (typical volume of about $10^{-3}\ mm^3$). Each domain contains billions of dipoles, all lined up. For iron each atom in a domain has a moment of around 2.2 Bohr magnetons [Schwarz]. But the domains themselves are randomly oriented and because of this random orientation the net magnetic field generated is usually 0 - that's why any piece of iron is not a permanent magnet. A sample of domains can be seen on figure 1.

When placing a piece of iron into a strong magnetic field, the torque $\mathbf{m} \times \mathbf{B}$ tends to align the dipoles parallel to the field. Since they like to stay parallel to their neighbors, most of the dipoles will resist the torque. However, at the boundary

[2]Based on [Griffiths].

between two domains, there are competing neighbors, and the torque will throw its weight on the side of the domain most nearly parallel to the field; this domain will win over some converts, at the expense of the less favorably oriented one. The net effect of the magnetic field, then, is to move the domain boundaries. Domains parallel to the field grow, and the others shrink. If the field is strong enough, one domain takes over entirely, and the iron is said to be "saturated".

When the field is turned off, there will be some return to randomly oriented domains, but it is far from complete - there remains a preponderance of domains in the original direction. This is the *hysteresis*.

One thing that can destroy uniform alignment of the spins is random thermal motions. This happens at a precise temperature called the *Curie point*[3] when a ferromagnetic behavior is abruptly changed into paramagnetic behavior. This abrupt change is known as a *second order phase transition*[4].

4 Monte Carlo Methods

Monte Carlo methods are based on the idea of repeated random sampling of the search space and the application of statistics to compute the searched value. Monte Carlo methods tend to be used when it is infeasible or impossible to compute an exact result with a deterministic algorithm.

The most famous example of a Monte Carlo method is probably the Buffon[5] needle [Hamming]. In 1773 Buffon observed that if a needle of length $L \leq 1$ were tossed at random onto a horizontal surface ruled with equally spaced lines, say at a unit spacing, then the probability of a needle crossing a line is

$$P = \frac{2L}{\pi}.$$

He reasoned, therefore, that he could experimentally determine the value of π by making repeated trials.

5 Markov Chain[6]

We can look at a system evolving from one state into another as a chain of states: $x_0 \rightarrow x_1 \rightarrow ... \rightarrow x_n$. We can state the probability of moving from one state into another ($x_{n-1} \rightarrow x_n$) as $P(x_n|x_{n-1}, x_{n-1}, ..., x_0)$ - that is the probability may depend on all the previous states of the system.

[3] Named after Pierre Curie (1859-1906), and refers to a characteristic property of a ferromagnetic or piezoelectric material. The curie point of iron is 768 $^{\circ}C$.

[4] Second-order phase transitions are continuous in the first derivative but exhibit discontinuity in a second derivative of the free energy.

[5] Named after Georges-Louis Leclerc, Comte de Buffon (1707-1788), French naturalist, mathematician, biologist, cosmologist and author.

[6] Based on [Heermann].

The *Markov process* or *Markov chain* is a system where the probability of a state moving to a specific next state ($x_{n-1} \to x_n$) only depends on the previous state x_{n-1}:

$$P(x_n|x_{n-1},...,x_0) = P(x_n|x_{n-1}).$$

The probability of a specific state change from x_0 to x_n is

$$\begin{aligned} P(x_0,...,x_n) &= P(x_n|x_{n-1})P(x_{n-1}|x_{n-2})...P(x_1|x_0)P(x_0) \\ &= P(x_n|x_{n-1})P(x_{n-1}|x_{n-2})...P(x_1|x_0)\cdot a_0 \end{aligned}$$

where a_0 is the probability of the starting state x_0.

What we are interested in is the probability distribution of the system's states. After a system starts from x_0 it should evolve and hopefully settle to a limited number of states with specific probabilities for each state at the given step - this is the probability distribution. More interesting is the case of an invariant probability distribution.

A probability distribution (u_k) is called *invariant* for a given Markov chain if it satisfies:

$$\begin{aligned} &\text{(i)} && \text{for all } k:\ u_k \geq 0, \\ &\text{(ii)} && \sum_k u_k = 1, \\ &\text{(iii)} && u_j = \sum_i u_i p_{ij}. \end{aligned}$$

The probability of a system moving from one state into another in n steps is denoted as $p_{ij}^{(n)}$.

A chain is *irreducible* if, and only if, every other state can be reached from every state. If the chain is reducible, the sequence of states will fall into classes with no transitions from one class into the other.

An example of a system with just four states with transition probabilities arranged into a stochastic matrix:

$$\begin{bmatrix} 1/2 & 1/4 & 0 & 1/4 \\ 0 & 1/3 & 2/3 & 0 \\ 0 & 1 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \end{bmatrix}.$$

The probability from going from state $1 \to 1$ is $1/2$, from state $1 \to 2$ is $1/4$, from state $1 \to 3$ is 0 and so on. This matrix is not irreducible - one can't go from state 2 to state 1 or 4. Once the system has reached the state 2 or 3 it is trapped.

A state x_i has a period $t > 1$ if $p_{ii}^{(n)} = 0$ unless $n = zt$ and t is the largest integer with this property. A state is *aperiodic* if no such $t > 1$ exists.

Let $f_{ij}^{(n)}$ denote the probability that in a process starting from x_i the first entry to x_j occurs at the n-th step. Further, let

$$\begin{aligned} f_{ij}^{(0)} &= 0, \\ f_{ij} &= \sum_{n=1}^{\infty} f_{ij}^{(n)}, \\ \mu_i &= \sum_{n=1}^{\infty} n f_{ii}^{(n)}. \end{aligned}$$

Then f_{ij} is the probability that starting from x_i the system will ever pass through x_j. In the case that $f_{ij} = 1$ the state x_i is called *persistent*, and μ_i is termed the *mean recurrence time*.

A state x_i is called *ergodic* if it is aperiodic and persistent with a finite mean recurrence time. A Markov chain with only ergodic elements is called ergodic. System's ergodicity basically means that any state is accessible from any other state. More strongly expressed, any state must be accessible from any other state in a finite number of transitions.

An irreducible aperiodic Markov chain possesses an invariant distribution if, and only if, it is ergodic. In this case $u_k > 0$ for all k and the absolute probabilities tend to u_k irrespective of the initial distribution.

6 The Metropolis Algorithm[7]

Suppose we want to calculate a property A of a system. All we need is a distribution function $P(x)$ that specifies how much the system spends in state x. The distribution $P(x)$ does not necessarily have to be a probability, that is it does not have to be normalized and so have a unit integral over the sampled region. Integrating over all the system states Ω we get

$$\langle A \rangle = \frac{1}{Z} \int_{\Omega} A(x) P(x) dx$$

where

$$Z = \int_{\Omega} P(x) dx$$

is the partition function used to normalize the distribution function $P(x)$. Here to calculate the property A we had to integrate over all the states Ω.

The Metropolis Monte Carlo algorithm does not sample a multidimensional region uniformly. Rather, the goal is to visit a point x with a probability proportional to the given distribution function $P(x)$. So the huge advantage of the Metropolis algorithm is avoiding the large search space and automatically concentrating

[7] Based on [NumRec, Heermann].

the search where $P(x)$ is large. As the sites visited are with the probability proportional to the distribution function, calculating the property we are interested in reduces to

$$\langle A \rangle = \frac{1}{n} \sum_{k=1}^{n} A(x_k)$$

where n is the number of steps taken.

Metropolis algorithm is based on two ideas. Firstly, the search is not done randomly but rather through an ergodic Markov chain so that in theory we could visit every possible state x. Using a Markov chain means the systems is stepped through a series of states that are in close proximity.

The second idea is to pick a transition function $W(x \to x')$ that satisfies the detailed balance equation

$$P(x)W(x \to x') = P(x')W(x' \to x)$$

where $W(x \to x')$ is the probability that a system transitions from state x to x'. This equation expresses the idea of physical equilibrium in the reversible transition

$$x \leftrightarrow x'$$

and is sufficient but not necessary condition for the Markov chain to converge ultimately to the desired distribution.

As the initial state may be far off from the searched equilibrium where $P(x)$ is large, the simulation may have to step through a number of steps at first before measurements can be taken. So this equilibration is an essential part of the algorithm.

To apply the Metropolis algorithm a transition function must be found that satisfies the detailed balance equation. How this is done for the Ising model can be seen from the next section.

7 The Ising Model

The Ising model was first conceived by Wilhelm Lenz[8], who suggested it as the Ph.D. topic for his graduate student Ernst Ising[9] in the 1920 [GiordNakan].

In the ferromagnetic Ising model we have a lattice of N discrete variables called *spins* that can have a value of only +1 or -1. Neighboring spins interact with each other through an exchange coupling. On the two-dimensional lattice in figure 2 each spin interacts with its four neighbors.

Boundaries must also be considered. If the lattice just terminates at the boundary the spins there have fewer spins to interact with and that can skew the results. To prevent that the most common method used is the *periodic boundary* - the spins at extremities are connected to each-other. For the two-dimensional Ising model it

[8] Wilhelm Lenz (1888-1957), German physicist.

[9] Ernst Ising (1900-1998), German physicist.

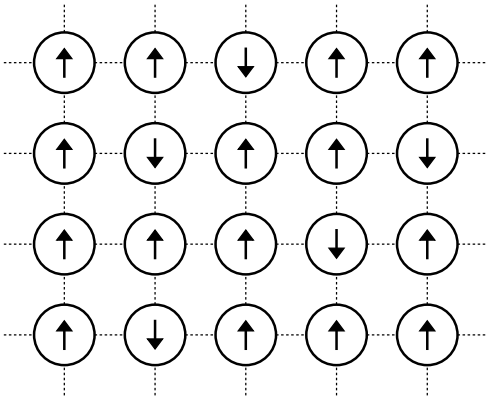

Figure 2: An example 2-dimensional Ising model spin configuration.

means that the spin lattice is closed and forms a torus. If the lattice is of sufficient size the periodic boundary itself will usually not cause any problems.

Let a specific spin configuration be known as state $x = (s_1, s_2, ..., s_N)$. The Hamiltonian (energy) of the system in state x is then:

$$\mathscr{H}(x) = -J \sum_{\langle i,j \rangle} s_i s_j - \mu H \sum_i s_i$$

where μ denotes the magnetic moment of a spin, H is the external magnetic field and J is the exchange coupling energy between neighboring cells. The sum $\sum_{\langle i,j \rangle} s_i s_j$ means that exchange energy is counted only for the neighbors of each spin.

If neighboring spins point in the same direction, then the energy contributed is $-J$ (assuming J is positive) but if they are anti-parallel, then $+J$. The system generally wants to go to lowest energy possible and so parallel spins are favored.

The probability for the system being in a specific state x represented by the Hamiltonian $\mathscr{H}(x)$ is proportional to

$$P(x) \propto e^{-\frac{\mathscr{H}(x)}{k_B T}}$$

where k_B is the Boltzmann constant and T is the temperature. This means that the system follows the Boltzmann energy distribution or in other words is a canonical ensemble. So temperature has a disorderly effect on the system causing it to diverge from the lowest possible energy.

Suppose we can change the system from state x to x' by toggling the value of spin s_i, so that $x' = (s_1, s_2, ..., -s_i, ..., s_N)$. To apply the Metropolis algorithm we need to develop a transition function. We can use a function where $W(x' \to x) = 1$ whenever energy is decreased going from state x' to x but in reverse it is

$$W(x \to x') = \frac{P(x')}{P(x)} = e^{-\left(\frac{\mathscr{H}(x')}{k_B T} - \frac{\mathscr{H}(x)}{k_B T}\right)} = e^{-\frac{\Delta \mathscr{H}}{k_B T}}$$

where $\Delta\mathscr{H}$ is the change in the energy of the system going from x to x'. It is easy to verify that this satisfies the detailed balance equation for the Metropolis algorithm and there is also no need to know the exact distribution function $P(x)$ as common terms are canceled out.

We can now describe a detailed Metropolis algorithm for the Ising model [Landau, GiordNakan, Heermann]:

1. Set the desired T and H.
2. Initialize all the spins in the system $x = (s_1, s_2, ..., s_N)$. We can take all spins up or completely random.
3. Perform a desired number of Monte Carlo sweeps through the lattice.
   (a) For a given sweep, loop through all the spins s_i in sequence:
      i. Generate a trial configuration x' by reversing the given spin's direction: $x' = (s_1, s_2, ..., -s_i, ..., s_N)$.
      ii. Calculate the energy $\mathscr{H}(x')$.
      iii. If $\mathscr{H}(x') \leq \mathscr{H}(x)$, accept the new configuration and set $x = x'$.
      iv. If $\mathscr{H}(x') > \mathscr{H}(x)$, accept with relative probability $P = e^{-\frac{\Delta\mathscr{H}}{k_B T}}$:
         A. Choose a uniform random number $R \in [0, 1]$.
         B. $x = \begin{cases} x' & \text{if } P \geq R \text{ (accept)} \\ x & \text{if } P < R \text{ (reject)} \end{cases}$
   (b) At the start perform a number of sweeps to reach the equilibrium.
   (c) Once the equilibrium is reached record the new energy, magnetization, and any other quantities of interest after each sweep.

The property we are interested in here is the magnetization. It can vary from -1 to $+1$ and is simply the weighted sum of all the spins:

$$m = \frac{1}{N} \sum_{k=1}^{N} s_k$$

8 Implementation

The Ising model was implemented in the C++ language on a Linux system using GCC. The implementation is three-dimensional Ising model with a periodic boundary, that is each spin has six neighbors to interact with. Following constants and parameters were used at the simulations:

- The bound exchange constant $J = 0.5$.
- Boltzmann constant $k_B = 1.0$.

- Spin values $\{-1.0, 1.0\}$.
- 3D spin lattice with size $LN \times LN \times LN$ where LN ranged from 10 ... 30.
- Temperature ranging from 0.0 ... 6.0.
- Magnetic field component μH ranging from -5.0 ... 5.0.
- Equilibration steps: 1000.
- Sampling steps: 1000.

At the start of the simulation all the spins are set up or down depending on the initial magnetic field. After that a number of steps are taken to reach an equilibrium and after that sampling steps can be taken during which different system properties are calculated. At each step all the spins were visited in sequence and toggled based on the Metropolis algorithm.

For schematics and visualizations the following tools were used: gnuplot, PovRay and xfig. The source code can be found in appendix A. Additionally, a web site with the code archive including a Makefile and visualization scripts has been set up at http://www.mare.ee/indrek/ising/.

9 Results

The simulation time dependence on the lattice size property LN can be seen on figure 3. As expected it responds at around LN^3.

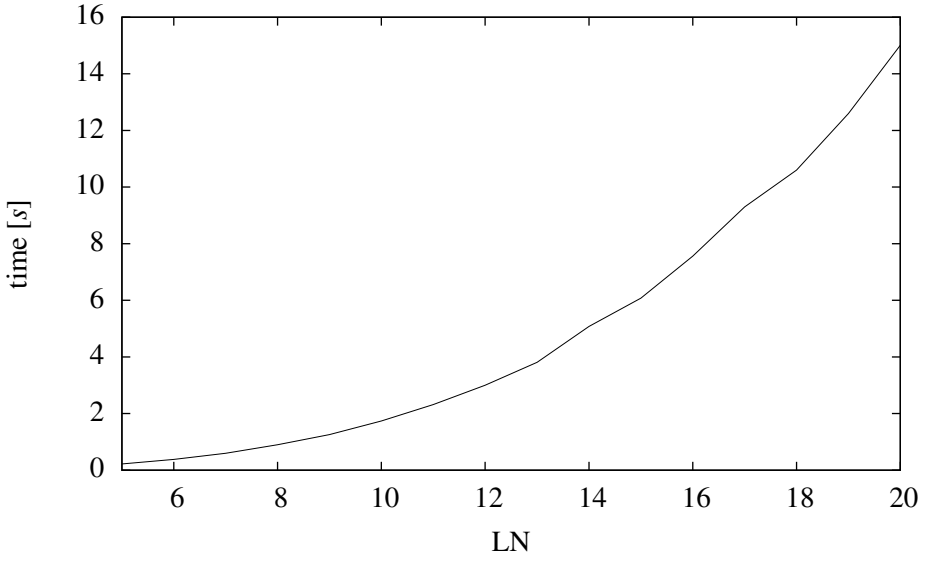

Figure 3: Relative running time $t = t(LN)$

The main interest of this simulation is the magnetization depending on temperature and where and how the second order phase transition from magnetism into paramagnetism happens. This can be seen on figure 4 with different lattice sizes.

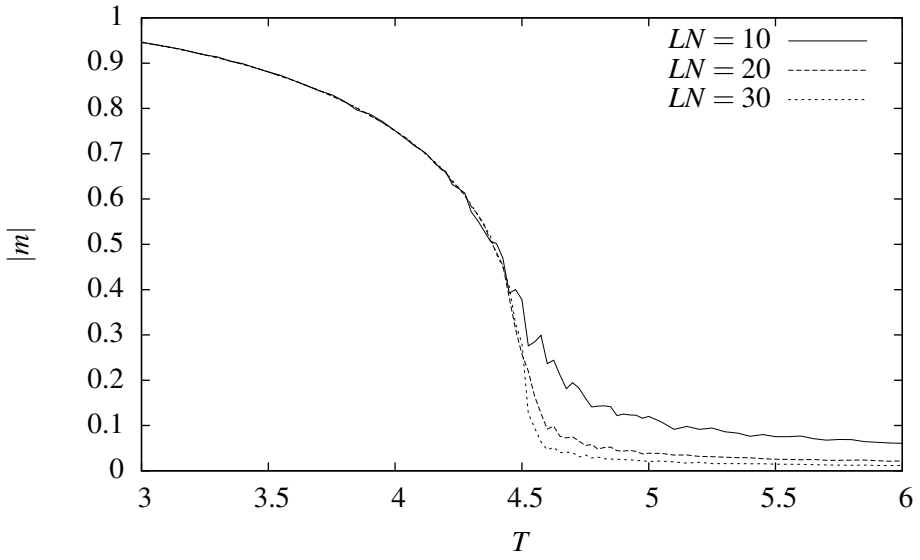

Figure 4: Magnetization $m = m(T)$ for different lattice sizes.

The hysteresis curve is also interesting to look at. In this simulation the magnetic field at the start was 0, then increased to $+5$, then decreased to -5 and then back to $+5$. The corresponding magnetization graphs can be seen on figure 5. At temperature $T = 0$ the external magnetic field at the given level had no effect on the magnetization direction. At temperature $T = 2.0$ we can see a clear hysteresis curve. It seems the Ising model does allow for some hysteresis effect.

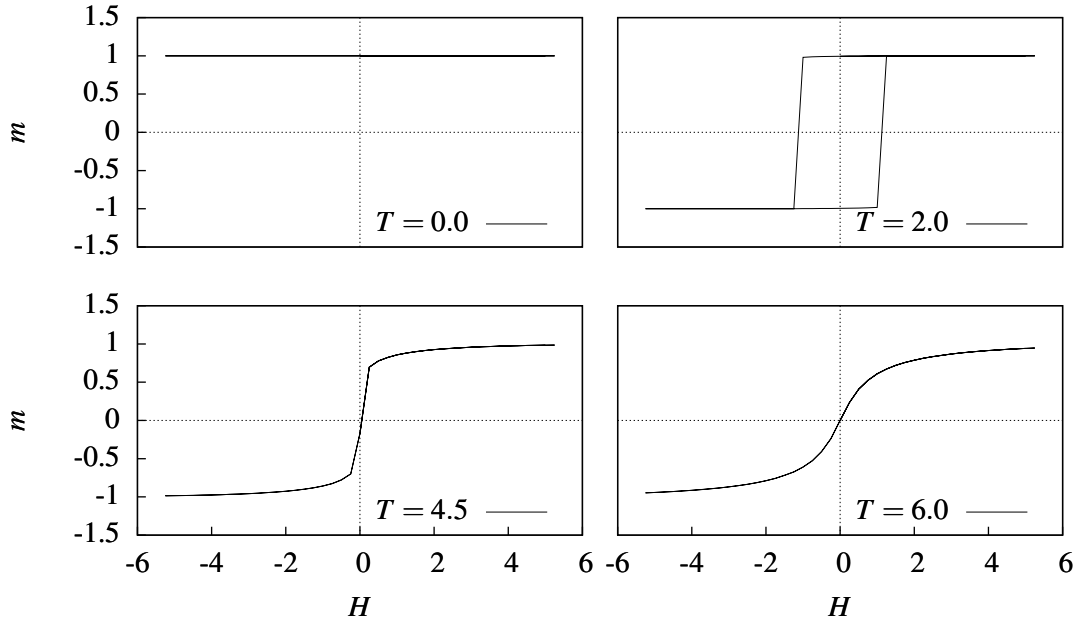

Figure 5: Magnetization hysteresis curves $m = m(H)$.

Finally an attempt was made to visualize the 3D lattice itself by cutting a quarter out of the lattice and visualizing the remainder using PovRay. The resulting image is on figure 6.

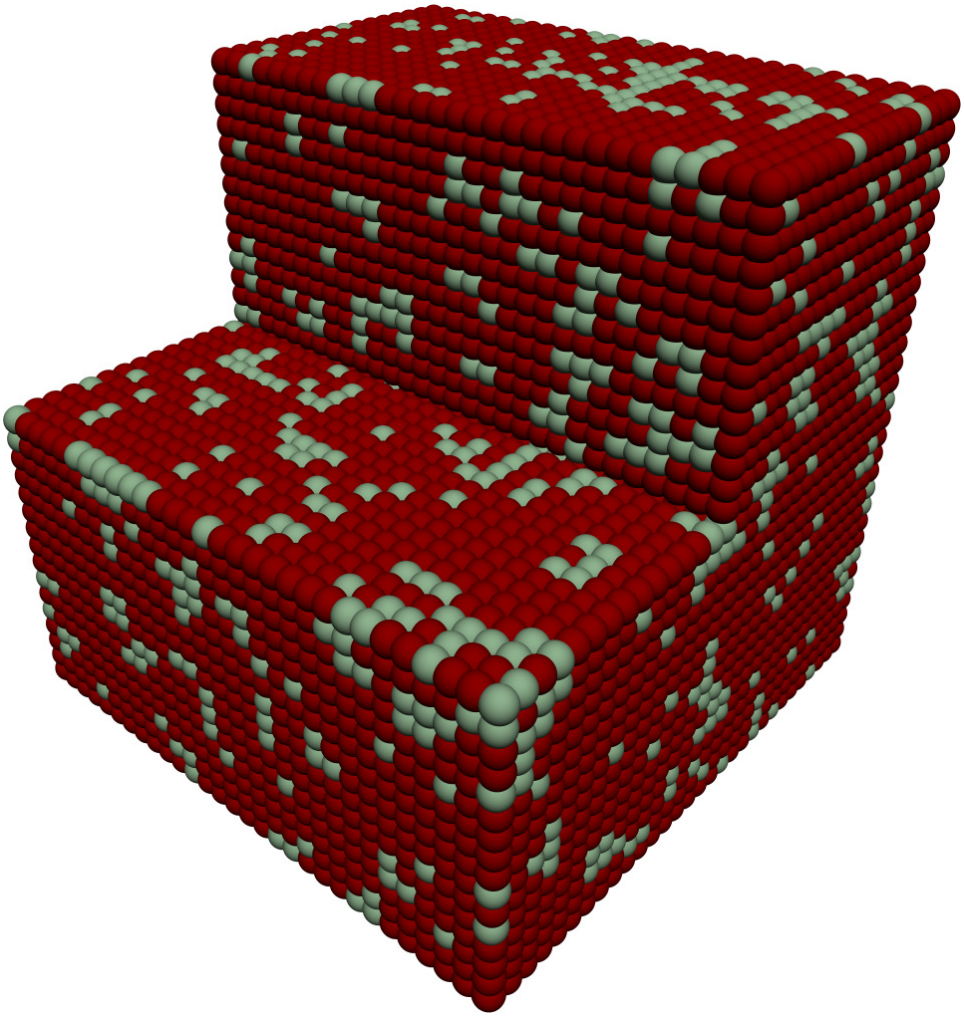

Figure 6: 30x30x30 spins lattice with $m = 0.58$ at $T = 4.3$.

References

[Purcell] Edward M. Purcell, "Electricity and Magnetism", second edition, 1985.

[Griffiths] David J. Griffiths, "Introduction to Electrodynamics", third edition, 1999.

[Schwarz] Melvin Schwartz, "Principles of Electrodynamics", 1987.

[Heermann] Dieter W. Heermann, "Computer Simulation Methods", second edition, 1990.

[NumRec] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, "Numerical Recipes: The Art of Scientific Computing", third edition, 2007.

[Hamming] R. W. Hamming, "Numerical Methods for Scientists and Engineers", second edition, 1973.

[GiordNakan] Nicholas J. Giordano, Hisao Nakanishi, "Computational Physics", second edition, 2006.

[Landau] Rubin H. Landau, Manuel J. Páez, Christian C. Bordeianu, "Computational Physics", second edition, 2007

A C++ Code

```
// Copyright (C) 2008 Indrek Mandre <indrek@mare.ee>

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>

#define JBOUND 0.5          // exchange bound energy
#define KB 1.0              // the Boltzmann's constant
#define TAU 1.0             // time scale factor

#define RNDSEED 1           // random seed

class Grid
{
  struct Cell
  {
    char v;   // spin value, -1 or +1
    char nc;  // neighbouring spins sum
  };

  int LN;
  double muH;
  Cell *_GRID;

  double TRMAP[14]; // transition probability map

  int ebs;          // energy bounds sum
  int sbalance;     // up/down spins sum

  double temp;  // temperature

  inline Cell& get (int plane, int row, int column)
  {
    if ( row == -1 ) row = LN - 1;
    if ( row == LN ) row = 0;
    if ( column == -1 ) column = LN - 1;
    if ( column == LN ) column = 0;
    if ( plane == -1 ) plane = LN - 1;
    if ( plane == LN ) plane = 0;
    return fget(plane, row, column);
  }

  inline Cell& fget (int plane, int row, int column)
  {
    return _GRID[plane * LN * LN  + row * LN + column];
  }

public:
  Grid (int LN = 16, double temp = 0, double muH = 0) :
      LN(LN), _GRID(0) { setup(LN, temp, muH); }
  ~Grid () { delete [] _GRID; }

  // set up the simulation at temperature t
  void setup (int LN = 16, double t = 0, double muH = 0)
  {
    this->LN = LN;
    this->muH = muH;
```

```
  delete [] _GRID;
  _GRID = new Cell [LN*LN*LN];
  temp = t;
  sbalance = 0;
  ebs = 0;

  // set all spins to alternating -1/1
  for ( int i = 0; i < LN; i++ )
      for ( int j = 0; j < LN; j++ )
          for ( int k = 0; k < LN; k++ )
              fget(i, j, k).v = muH < 0 ? -1 : 1;

  // calculate contributions from neighbours
  for ( int i = 0; i < LN; i++ )
      for ( int j = 0; j < LN; j++ )
          for ( int k = 0; k < LN; k++ )
              fget(i, j, k).nc =
                  get(i - 1, j, k).v + get(i + 1, j, k).v +
                  get(i, j - 1, k).v + get(i, j + 1, k).v +
                  get(i, j, k - 1).v + get(i, j, k + 1).v;

  // calculate energy bound count
  for ( int i = 0; i < LN; i++ )
      for ( int j = 0; j < LN; j++ )
          for ( int k = 0; k < LN; k++ )
            {
              Cell& c = fget(i, j, k);
              ebs += c.v * c.nc;
              sbalance += c.v;
            }

  reprob ();
}

// calculate probabilities for the state transitions
void reprob ()
{
  for ( int ncv = -6; ncv <= 6; ncv += 2 )
    {
      int i = ncv + 6;
      // from the positive spin +1
      double dH = -4 * (-JBOUND * ncv) + 2 * muH;
      double prob = exp(-dH/(KB * temp)) / TAU;
      if ( prob > 1.0 / TAU ) prob = 1.0 / TAU;
      TRMAP[i] = prob;

      // from the negative spin -1
      dH = -4 * (-JBOUND * ncv) - 2 * muH;
      prob = exp(-dH/(KB * temp)) / TAU;
      if ( prob > 1.0 / TAU ) prob = 1.0 / TAU;
      TRMAP[i + 1] = prob;
    }
}

// change the muH parameter
void set_muH(double muH)
{
  this->muH = muH;
  reprob ();
}

// toggle the spin of the given cell
```

```
inline void toggle (int plane, int row, int column)
{
  Cell& c = get(plane, row, column);
  c.v *= -1;
  get(plane - 1, row, column).nc += 2 * c.v;
  get(plane + 1, row, column).nc += 2 * c.v;
  get(plane, row - 1, column).nc += 2 * c.v;
  get(plane, row + 1, column).nc += 2 * c.v;
  get(plane, row, column - 1).nc += 2 * c.v;
  get(plane, row, column + 1).nc += 2 * c.v;
  sbalance += 2 * c.v;
  ebs += 4 * c.v * c.nc;
}

// move simulation on by one simulation step
void step ()
{
  for ( int i = 0; i < LN; i++ )
      for ( int j = 0; j < LN; j++ )
          for ( int k = 0; k < LN; k++ )
            {
              Cell& c = fget(i, j, k);
              if ( drand48() < TRMAP[c.v * c.nc + 6 +
                  (c.v == -1 ? 1 : 0) ] ) toggle (i, j, k);
            }
}

// calculate the energy in the simulation
double energy ()
{
  return -JBOUND * ebs - muH * sbalance;
}

// calculate the energy contribution from the given cell
double energy (int plane, int row, int column)
{
  Cell& c = get(plane, row, column);
  return -JBOUND * c.v * c.nc - muH * c.v;
}

inline int spin (int plane, int row, int column)
{
  return get(plane, row, column).v;
}

double magnetization ()
{
  return (double)sbalance / (LN * LN * LN);
}

struct sample_data
{
  double magnetization;
  double abs_magnetization;
};

void sample (sample_data& out, size_t eqc, size_t sampc)
{
  for ( size_t i = 0; i < eqc; i++ )
      step();
  double m = 0;
  double abs_m = 0;
```

```
    for ( size_t i = 0; i < sampc; i++ )
      {
        step();
        m += magnetization();
        abs_m += fabs(magnetization());
      }
    out.magnetization = m / sampc;
    out.abs_magnetization = abs_m / sampc;
  }
};

static void compare_grid(int n)
{
  printf ("Compare grid %d\n", n);
  char fn[128];
  sprintf (fn, "cgrid%d.txt", n);
  FILE *fp = fopen (fn, "w+");
  Grid *grid = new Grid ();
  for ( double t = 3; t < 6.01;
      t += (fabs(t - 4.5) < 0.5) ? 0.025 : 0.05 )
    {
      Grid::sample_data data;
      grid->setup(n, t);
      grid->sample (data, 1000, 1000);
      fprintf (fp, "%f %f\n", t, data.abs_magnetization);
    }
  delete grid;
  fclose (fp);
}

static void compare_hm (double temp)
{
  printf ("M=M(H) / TEMP = %.1f\n", temp);
  char fn[128];
  sprintf (fn, "chm%.1f.txt", temp);
  FILE *fp = fopen (fn, "w+");
  double h = 0;
  Grid *grid = new Grid (20, temp, h);
  Grid::sample_data data;
  grid->sample (data, 1000, 0);
  for ( ; h <= 5.01; h += 0.25 )
    {
      grid->set_muH (h);
      grid->sample (data, 200, 200);
      fprintf (fp, "%f %f\n", h, data.magnetization);
    }
  for ( ; h >= -5.01; h -= 0.25 )
    {
      grid->set_muH (h);
      grid->sample (data, 200, 200);
      fprintf (fp, "%f %f\n", h, data.magnetization);
    }
  for ( ; h <= 5.01; h += 0.25 )
    {
      grid->set_muH (h);
      grid->sample (data, 200, 200);
      fprintf (fp, "%f %f\n", h, data.magnetization);
    }
  delete grid;
  fclose (fp);
}
```

```
static void running_time (int n1, int n2)
{
  printf ("Running time...\n");
  FILE *fp = fopen ("rtime.txt", "w+");
  Grid *grid = new Grid ();
  for ( int i = n1; i <= n2; i++ )
    {
      printf ("running_time: %d\n", i);
      Grid::sample_data data;
      grid->setup (i, 0);
      clock_t t1, t2;
      t1 = clock ();
      grid->sample (data, 100000, 0);
      t2 = clock ();
      fprintf (fp, "%d %f\n", i, (double)(t2 - t1)/1000000.0);
    }
  delete grid;
  fclose (fp);
}

static void slice (double temp)
{
  printf ("Running slice...\n");
  FILE *fp = fopen ("slice.inc", "w+");
  Grid *grid = new Grid (30, temp);
  Grid::sample_data data;
  grid->sample (data, 2000, 1000);
  printf ("Magnetization: %f\n", data.magnetization);
  for ( int i = 0; i < 30; i++ )
      for ( int j = 0; j < 30; j++ )
          for ( int k = 0; k < 30; k++ )
            {
              if ( k <= 15 || j <= 15 )
                  fprintf (fp, "sphere{<%d,%d,%d>,0.7 texture{%s}}\n",
                      i, j, k, grid->spin(i, j, k) < 0 ? "SD" : "SU");
            }
  delete grid;
  fclose (fp);
}

int main ()
{
  srand48 (RNDSEED);

  slice (4.3);

  compare_hm (0.0);
  compare_hm (2.0);
  compare_hm (4.5);
  compare_hm (6.0);

  running_time (5, 20);

  compare_grid (10);
  compare_grid (20);
  compare_grid (30);

  return 0;
}
```