Some Property of Ising Model

Yongle Bai 202011150087

1 The detailed setup of the problem

The Ising model (or Lenz-Ising model or Ising-Lenz model), named after the physicists Ernst Ising and Wilhelm Lenz, is a mathematical model of ferromagnetism in statistical mechanics. The model consists of discrete variables that represent magnetic dipole moments of atomic "spins" that can be in one of two states (+1 or 1). The spins are arranged in a graph, usually a lattice (where the local structure repeats periodically in all directions), allowing each spin to interact with its neighbors. Neighboring spins that agree have a lower energy than those that disagree; the system tends to the lowest energy but heat disturbs this tendency, thus creating the possibility of different structural phases. The model allows the identification of phase transitions as a simplified model of reality. The two-dimensional square-lattice Ising model is one of the simplest statistical models to show a phase transition.

Now we consider a 2-dimensional lattice with period boundary and size $n \times n$. We use $\Lambda := \mathbb{Z}_n \times \mathbb{Z}_n$ as the index set. Let $\sigma = \{\sigma_k\}_{k\in\Lambda}$ be the state of the lattice. The direction of each spin in the lattice is regarded as an element in σ, and ± 1 represent two direction. For any two adjacent sites $i, j \in \Lambda$ there is a spin-spin interaction J. Also, a site $j \in \Lambda$ has an external magnetic field H interacting with it. The energy of a configuration σ is given by the Hamiltonian function

$$H(\sigma) = -\sum_{i,j\in\Lambda, i\sim j} J\sigma_i\sigma_j - \sum_{i\in\Lambda} H\sigma_i \tag{1}$$

where $i \sim j$ means i and j is adjacent. The configuration probability is given by the Boltzmann distribution with inverse temperature $\beta \geq 0$:

$$\mathbb{P}(\sigma) = \frac{e^{-\beta H(\sigma)}}{Z},$$

where $\beta = 1/(k_B T)$, and the normalization constant

$$Z = \sum_{\sigma} e^{-\beta H(\sigma)}$$

is the partition function.

Our aim is to find out how does the temperature influence the property of lattice and determine the critical temperature. In this article, we will use Metropolis sampling to solve following two problems:

Problem 1. *Take $J = 1, k_B = 1, H = 0$, let N be the size of the lattice and T be the temperature. Calculate internal energy u in the process:*

$$u = \frac{U}{N^2}, U = \mathbb{E}(H) = \sum_{\sigma \in 2^{\Lambda}} \mathbb{P}(\sigma) H(\sigma) \tag{2}$$

and specific heat c:

$$c = \frac{C}{N^2}, C = \frac{k_B}{T^2} \mathbb{V}\mathrm{ar}(H) = \frac{k_B}{T^2} (\mathbb{E}(H^2) - \mathbb{E}(H)^2) \tag{3}$$

where $\mathbb{V}\mathrm{ar}$ is the variance. And plot graph of $u - T, c - T$. And check the temperature near the critical temperature $T_c = \frac{2|J|}{k_B \ln(1+\sqrt{2})}$.

Problem 2. *Take $J = 1, k_B = 1, H \neq 0$. Fixing T, H, Calculate magnetization m in the process:*

$$m = \frac{M}{N^2}, M = \mathbb{E}(\sum_{i \in \Lambda} \sigma_i). \tag{4}$$

And plot graph of $m - (T, H)$. You may find some interesting result near $T = T_c$.

2 The procedure you take to do the computation

Whether to accept the convert of a spin is decided by the change of Hamiltonian before and after the convert, write ΔH, and temperature T. If $\Delta H < 0$, then accept the convert. If $\Delta H > 0$, then we accept the convert with probability of $e^{\frac{-\Delta H}{\beta}}$. After the convert of the spin, a convert will happen afterward. The procedure of the computation is following:

1. First, set the initial parameters: Set up the grid to imitate the magnet. Choose a state X_1 randomly in 2^Λ.

2. Secondly, we randomly pick a point in X_n to try to change its direction of magnet, and calculate the changed energy ΔH. And then, judge whether to accept the direction change of this point. If accepted, then the point will change direction; else, it will remain the old direction. Then we get X_{n+1}.

3. Repeat the process above for a long time, write Y.

4. Use $\mathbb{E}'(f) := \sum_{t=X}^{Y} f(X_t)$ to estimate $\mathbb{E}(f)$, to calculate c, u and m, where X is a certain number chose between 0 and Y.

5. Last, plot the graphs and analysis the results.

3 Analysis of the numerical results

Solution 1. *In Problem 1, we let T ranges in $0.5 \sim 4$ and convert 1000000 steps. In Figure 1 we can see as temperature become higher and higher, the value of the average Hamiltonian, u, become bigger and bigger. That's means high temperature will supply more energy for the spins to get a state with high Hamiltonian. Moreover, we can find that the growth rate of u is highest near $T = T_c$. When T is far from T_c, the value of u is a virtual constant.*

In Figure 2 we can see that the more temperature T go near T_c, the higher the specific heat is. Since the specific heat c is related to the variance of the Hamiltonian, we know that when T is near T_c, the Hamiltonian will be very unstable. In fact, when the temperature is low, the spins will finally get into the same direction, so the Hamiltonian will be very small; when the temperature is high, the lattice will get very mess, every spin will get up or down randomly, so the Hamiltonian will be very big; when the temperature is very near to T_c, the lattice will behave between them, thus the value of Hamiltonian will be big or small from time to time. So when $T = T_c$ the variance of Hamiltonian will be very big.

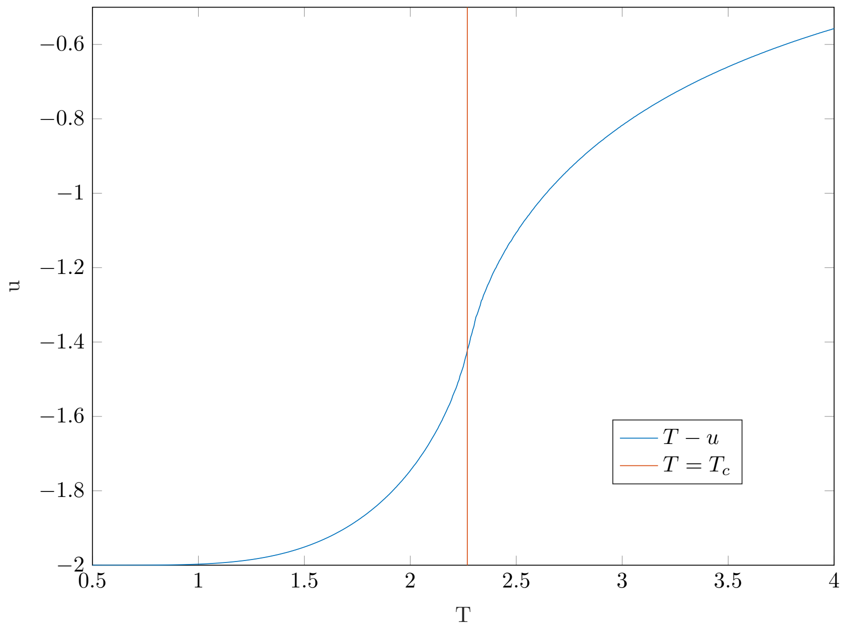

Figure 1: T-u

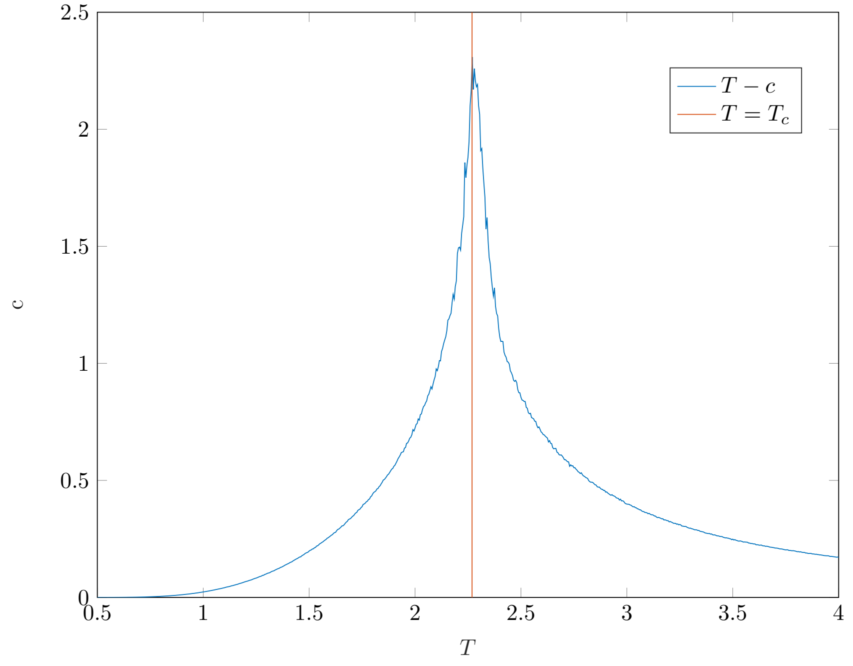

Figure 2: T-c

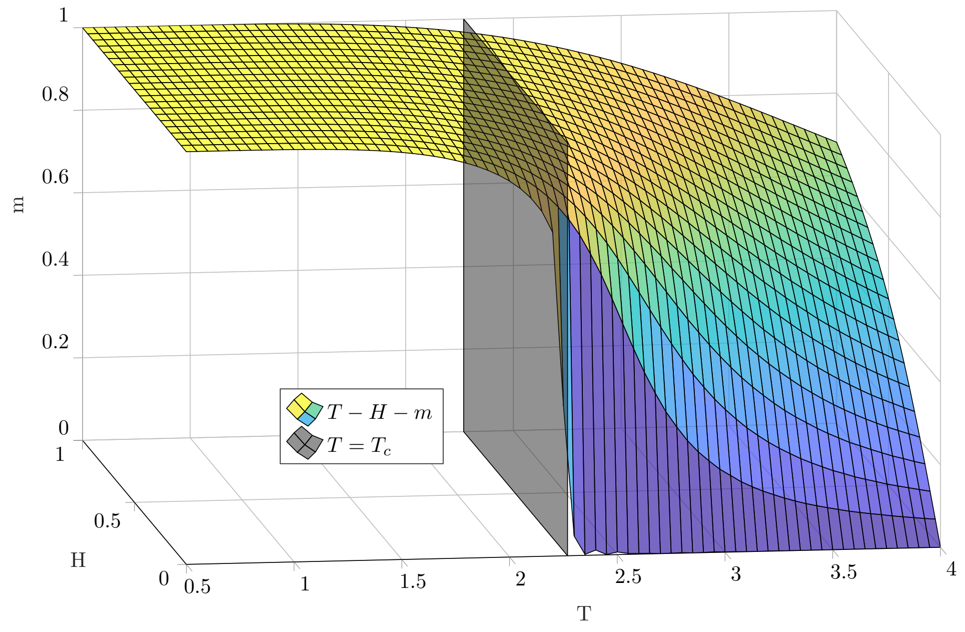

Figure 3: T-H-m

Solution 2. *We let T range from 0.5 to 4 and H range from 0 to 1 and run the program 1000000 steps, then compute the value of m for every pair of (T, H). From Figure 3 we can find that when $T < T_c$, the value of m will be near 1 not matter how small H is; when $T > T_c$, the value of m just increasing follow H: when H is small, m is near 0; when H is big, m also is big but can't reach 1.*

We can imagine, if this lattice is a magnet, then when the temperature is low, this magnet can easily be magnetized; when the temperature is high, this magnet can't be permanently magnetized.

4 The issues you encounter and how you overcome

The program cost several weeks every run, but we can't let our computer work so long. So we used our teacher's computer as server, and learned how to use ssh and sftp to work with a linux server.

5 Possible discussion about the results and further thinking

Ising model can not only work on a lattice but also work on a graph. The value of J, H also can change by the index. Using Ising on a general graph can help to solve the graph maximum cut problem. The graph maximum cut problem is set up as following:

Let G be a weighted undirected graph. A subset S of the vertex set $V(G)$ of G determines a cut of the graph G into S and its complementary subset $G \setminus S$. The size of the cut is the sum of the weights of the edges between S and $G \setminus S$. A maximum cut size is at least the size of any other cut, varying S. For the Ising model without an external field on a graph G, the Hamiltonian becomes the following sum over the graph edges $E(G)$.

$$H(\sigma) = -\sum_{ij\in E(G)} J_{ij}\sigma_i\sigma_j \tag{5}$$

Here each vertex i of the graph is a spin site that takes a spin value $\sigma_i = \pm 1$, and J_{ij} is the interaction of σ_i, σ_j. A given spin configuration σ partitions the set of vertices $V(G)$ into two σ-depended subsets, those with spin up V^+ and those with spin down V^-. We denote by $\delta(V^+)$ the σ-depended set of edges that connects the two complementary vertex subsets V^+ and V^-. The size $|\delta(V^+)|$ of the cut $\delta(V^+)$ to bipartite the weighted undirected graph G can be defined as

$$\left|\delta(V^+)\right| = \frac{1}{2}\sum_{ij\in\delta(V^+)} W_{ij} \tag{6}$$

where W_{ij} denotes a weight of the edge ij and the scaling $\frac{1}{2}$ is introduced to compensate for double counting the same weights $W_{ij} = W_{ji}$.

The identities

$$\begin{aligned} H(\sigma) &= -\sum_{ij\in E(V^+)} J_{ij} - \sum_{ij\in E(V^-)} J_{ij} + \sum_{ij\in\delta(V^+)} J_{ij} \\ &= -\sum_{ij\in E(G)} J_{ij} + 2\sum_{ij\in\delta(V^+)} J_{ij} \end{aligned} \tag{7}$$

where the total sum in the first term does not depend on σ, imply that minimizing $H(\sigma)$ in σ is equivalent to minimizing $\sum_{ij\in\delta(V^+)} J_{ij}$. Defining the edge weight $W_{ij} = -J_{ij}$ thus turns the Ising problem without an external field into a graph Max-Cut problem [4] maximizing the cut size $|\delta(V^+)|$, which is related to the Ising Hamiltonian as follows,

$$H(\sigma) = \sum_{ij\in E(G)} W_{ij} - 4\left|\delta(V^+)\right|. \tag{8}$$

Appendix

1 Vedios

Following videos only can be played with pdf viewer supporting javascript, such as AdobeReader or Okular. Or you can play the avi file with any video player.

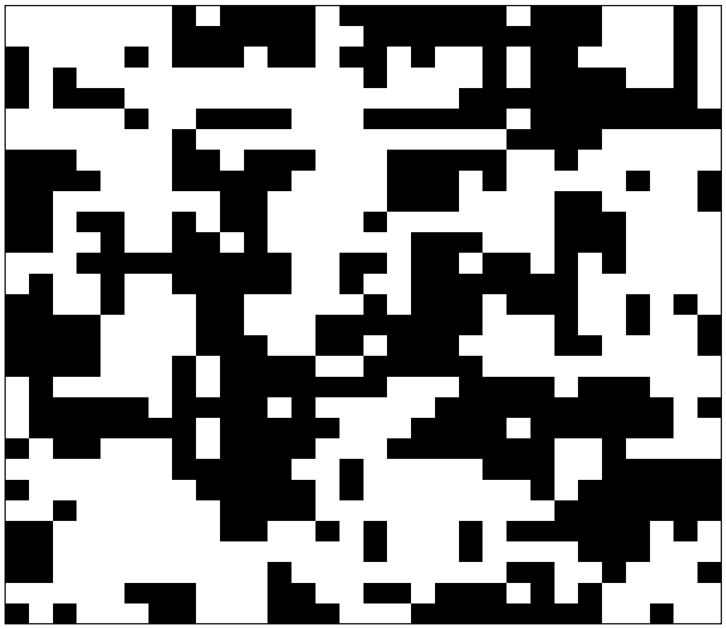

Figure 4: T=Tc

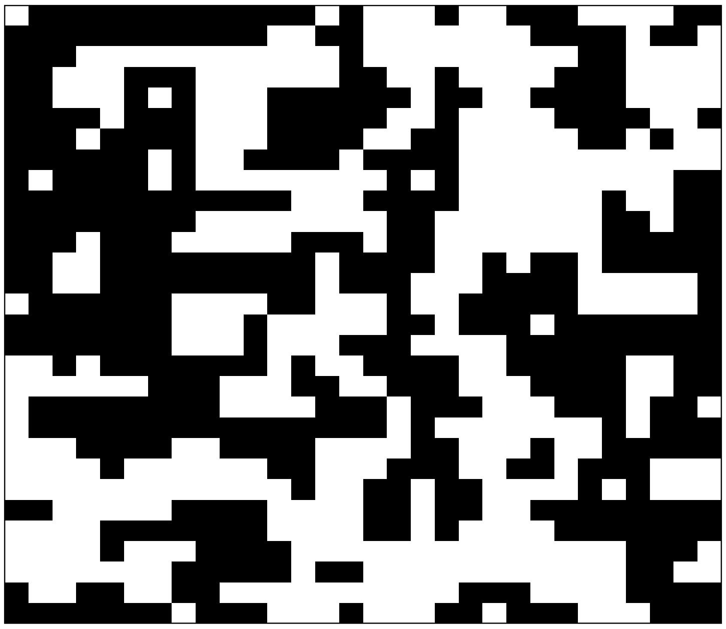

Figure 5: T=1

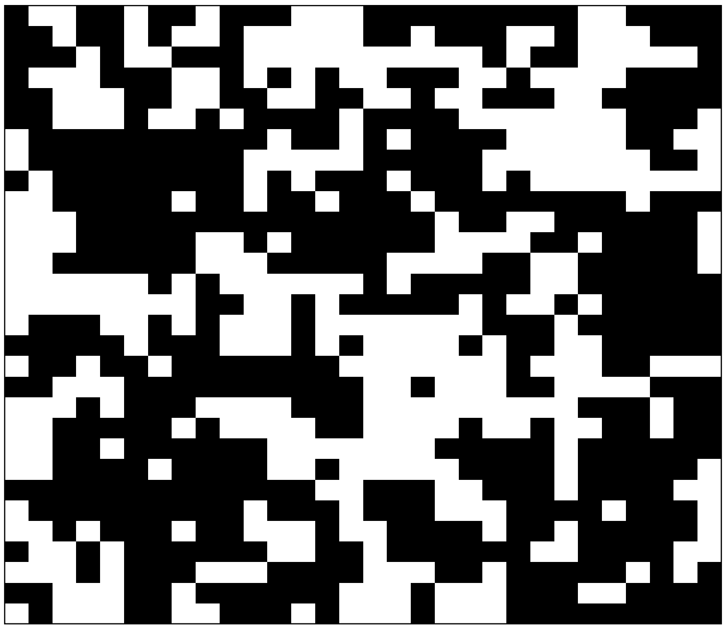

Figure 6: T=3

2 Codes

2.1 main

```
%% This program is used to make a 2D-Model of Ising.
%% Initialize:
%  Thread is the number of experiments
%  Kb is the Boltzmann constant.
Kb = 1;
%  Size is the size of grid.
Size = 60;
Size2 = Size * Size;
%  Times is the number of steps.
Times = 100000 * Size2;
%  J is the spin-spin interaction
J = 1;
%  Accepttime: we only use energy when time geq
   Accepttime
Accepttime = floor(Times * 0.8);
%  sumH2: The sum of Energy^2.
sumH2 = zeros(1,1,Thread);
%  sumH: The sum of Energy.
sumH = zeros(1,1,Thread);
%  sumMag: The sum of Mag.
sumMag = zeros(1,1,Thread);
%
%% Step 1: Generate a grid.
grid = GenerateGrid(Size,Thread);
%  Magnet: The internal Magnet of grid.
Magnet = sum(sum(grid));
%% Step 2: Initiallize. caculate energy,
Energy = getEnergyOfGrid(Size, grid, J, H);
%% Step 3:
for time = 1:Times
  %% Step 3.1: Choose a point randomly.
  coordinate = ceil(rand(2,1,Thread) .* Size);
  %% Step 3.2 Caculate the DeltaEnergy.
  DeltaEnergy = getDeltaEnergy(Size, grid, J, H,
     coordinate);
```

```matlab
  %% Step 3.3: judge whether to accept the change or not.
  Flag = WhetherAccept(DeltaEnergy, Kb, T,Thread);
  %% Step 3.4: If accept, change the grid and energy.
  Energy = Energy + DeltaEnergy .* Flag;
  Magnet = Magnet - 2 .* Flag .* grid(coordinate(1),coordinate(2),:);
  Flag = Flag * 2 - ones(1,1,Thread);
  Flag = - Flag;
  grid(coordinate(1),coordinate(2),:) = grid(coordinate(1),coordinate(2),:) .* Flag;
  %% Step 3.5: sum Z and uZ.
  if time > Accepttime
    sumH = sumH + Energy;
    sumH2 = sumH2 + Energy .* Energy;
    sumMag = sumMag + Magnet;
  end
  %% Step 3.5: plot picture of grid.
  if mod(time,floor(Times/100))==0
    % show the progress
    time/Times
  end
end
```

2.2 GenerateGrid

```matlab
function grid = GenerateGrid(Size,Thread)
grid = rand(Size,Size,Thread);
grid = grid > 0.5;
grid = 2 * grid - ones(Size,Size,Thread);
end
```

2.3 WhetherAccept

```matlab
%% This function is to judge whether to accept the change of a point.
```

```
% Input  DeltaEnergy: The change of the energy
%                  T: the temprature
%                 Kb: the Boltzmann constant.
% Outpur        Flag: 1 is accept, 0 is not accept.
function Flag = WhetherAccept(DeltaEnergy, Kb, T, Thread)
 Flag = DeltaEnergy <= 0 | rand(1,1,Thread) < exp(-DeltaEnergy ./ (Kb .* T));
end
```

2.4 getDeltaEnergy

```
%% This function is to caculate the change of energy after convert a point.
% Input  grid: the grid before change
%           J: the spin-spin interaction
%           H: the external field
%        Size: the size of the grid
%        coordinate: the coordinate of the converted point.
% Output  DeltaEnergy: The change of the energy.

function DeltaEnergy = getDeltaEnergy(Size, grid, J, H, coordinate)
%% Step 1: caculate the change of external energy
 DeltaExternalEnergy = 2 * H .* grid(coordinate(1),coordinate(2),:);
%% Step 2: caculate the change of interaction energy
 SurroundMagnet = grid(mod(coordinate(1),Size)+1,coordinate(2),:) + ...
                  grid(coordinate(1),mod(coordinate(2),Size)+1,:) + ...
                  grid(mod(coordinate(1)-2,Size)+1,coordinate(2),:) + ...
                  grid(coordinate(1),mod(coordinate(2)-2,Size)+1,:);
 DeltaInternalEnergy = 2 * J * SurroundMagnet .* grid(coordinate(1),coordinate(2),:);
```

```matlab
 %% Step 3: caculate the total change of energy
 DeltaEnergy = DeltaExternalEnergy +
    DeltaInternalEnergy;
end
```

2.5 getEnergyOfGrid

```matlab
%% this function is to caculate the energy of a grid
   .
% Input: grid: the grid
%           J: the spin-spin interaction
%           H: the external field
%           Size: the size of the grid
% Output: Energy: energy of the grid


function Energy = getEnergyOfGrid(Size, grid, J, H)
 %% Step 1: Caculate the external field energy
 ExternalEnergy = -H .* sum(sum(grid));
 %% Step 2: Caculate the interaction energy
 Leftgrid = [grid(:,2:Size,:),grid(:,1,:)];
 Upgrid = [grid(2:Size,:,:);grid(1,:,:)];
 InterEnergy = -J*sum(sum(Leftgrid .* grid + Upgrid
    .* grid));
 %% Step 3: Caculate the total energy
 Energy = ExternalEnergy + InterEnergy;
end
```

2.6 pro1

```matlab
clear;
%  T is the temprature, the third dimension
   represents different expenriment.
%  H is external field
T = 0.5:0.01:4;
Thread = size(T,2);
```

```
H = zeros(Thread,1);
T = permute(T,[1,3,2]);
H = permute(H,[3,2,1]);
main;
%% Step 4: caculate the averange internal energy, u,
   and the specific heat, c.
U = sumH ./ (Times - Accepttime);
C = Kb ./ T ./ T .* (sumH2 ./ (Times - Accepttime) -
   U .* U);
u = U ./ Size2;
c = C ./ Size2;
u = permute(u,[3,2,1]);
T = permute(T,[3,2,1]);
c = permute(c,[3,2,1]);
%% Step 5: plot picture.
plot(T,u);
saveas(gcf,"Tu.fig");
plot(T,c);
saveas(gcf,"Tc.fig");
```

2.7 pro2

```
clear;
%  T is the temprature, the third dimension
   represents different expenriment.
%  H is external field
% [T,H] = meshgrid(0.5:0.1:4,0:1:0);
[T,H] = meshgrid(1:2,0.2:0.1:0.4);
%  Because I only prepared one dimension for thread,
   so I need to make T and H one-dimentional.
X = size(T,1);
Y = size(T,2);
T = T(:);
H = H(:);
T = permute(T,[3,2,1]);
H = permute(H,[3,2,1]);
Thread = size(T,3);
main;
```

```
%% Step 4: caculate the averange magnet, m.
M = sumMag ./ (Times - Accepttime);
m = M ./ Size2;
%% Step 5: plot picture.
m = permute(m,[3,2,1]);
% H = permute(H,[3,2,1]);
% restore T and H as matrix
T = reshape(T,X,Y);
H = reshape(H,X,Y);
m = reshape(m,X,Y);
surf(T,H,m);
saveas(gcf,"THM.fig");
```

nparams=0
nups=0
linedefined=−1
what=C
short_src=[C]
istailcall=false
func=function: 0x4b9490
isvararg=true
namewhat=
currentline=−1
lastlinedefined=−1
source==[C]
=[C]