Chapter 6
The Ising Model

From the book:
Computer Simulations with Mathematica:
Explorations in Complex Physical and Biological Systems

by
Richard J. Gaylord
and
Paul R. Wellin

Introduction

The Ising system serves as one of the simplest models of interacting bodies in statistical physics. It has been used to model ferromagnets, anti-ferromagnetism and phase separation in binary alloys. The model has also been applied to spin glasses and neural networks. It has even been suggested that the Ising model is relevant to imitative behavior in general, including such disparate systems as flying birds, swimming fish, flashing fireflies, beating heart cells, spreading diseases, and even fashion fads.

The Probabilistic Ising Model

The probabilistic Ising model employs a constant temperature condition (known as the canonical ensemble formulation). The two-dimensional version of the model consists of an n x n square lattice in which each lattice site has associated with it a value of 1 (known as an *up spin*) or -1 (known as a *down spin*). Spins on adjacent, nearest-neighbor lattice sites (henceforth called neighbors) interact in a pair-wise manner with a strength J (known as the exchange energy or constant). When J is positive, the energy is lower when spins are in the same direction and when J is negative, the energy is lower when spins are in opposite directions. There may also be an external field of strength B (known as the magnetic field). The *magnetization* of the system is the difference between the number of up and down spins on the lattice. In the spin flipping process, a lattice site is randomly selected and it is either flipped (the sign of its value is changed) or not, based on the energy change in the system that would result from the flip, using The Metropolis method.

■ The Algorithm

(1) Create an n 'n by n' lattice consisting of randomly chosen site values of +1 and -1.

The following sequence of steps 2-4 will be executed a number of times (this is described in step 5), first using the initial lattice configuration, and then using the lattice configuration resulting from the previous run-through of the sequence. We will describe the steps in terms of an arbitrary lattice configuration, called lat.

(2) Select a random lattice site in lat.

(3) Determine the energy change involved in 'flipping' the spin at the selected lattice site. This is done in a number of steps:

(3a) Determine the neighbors to the selected site. When the selected site is in the interior of lat, the neighbors are the sites north (above), south (below), west (left) and east (right) of the site. When the selected site is along the border of lat, some neighbors are taken from the opposing side of the lattice.

note: This way of choosing neighbors for border sites is known as the reflecting or periodic boundary condition.

(3b) The energy change that would result from flipping the spin of the selected lattice site is determined by the quantity, 2 (value of selected site) (B + J (total spin from neighbors)), where B and J are input values.

(4) Use the Metropolis method to decide whether to flip the spin of the selected lattice site as follows:

(4a) Check if there is a negative energy change as a result of the flip.

(4b) If the energy change is non-negative, check if the exponential of (- energy change) is greater than a random number between 0 and 1.

(4c) If one of these conditions is satisfied, flip the spin.

(5) Execute the sequence of steps 2-4 m times.

(6) Create a sublist, monteCarloStepLis, containing every (n^2)th element from the list of lat configurations.

note: The use of every (n^2)th element corresponds to giving each lattice site an equal chance to be selected. Each element in monteCarloStepLis is said to correspond to one monte carlo step.

(7) Calculate for each element in monteCarloStepLis, some global property of the lattice, such as the long range order (the absolute value of the magnetization of the lattice).

■ Implementation

(1) An initial 'n by n' lattice of randomly oriented spins is created using

```
2 Table[Random[Integer], {n}, {n}] - 1
```

```
n = 4; lat = 2 Table[Random[Integer], {n}, {n}] - 1; MatrixForm[lat]
```

$$\begin{pmatrix} 1 & 1 & -1 & 1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 \end{pmatrix}$$

(2) A lattice site is randomly selected using

```
{i1, i2} = {Random[Integer, {1, n}], Random[Integer, {1, n}]}
```

```
{i1, i2} = {Random[Integer, {1, n}], Random[Integer, {1, n}]}
```

```
{2, 3}
```

(3) The energy change involved in 'flipping' the spin at the selected lattice site is calculated in a number of steps:

(3a) The neighbors to the selected site are lat[[dn, i2]], lat[[up, i2]], lat[[i1, rt]] and lat[[i1, lt]] where lat[[dn, i2]] is the lattice site beneath the site, lat[[up, i2]] is the lattice site above the site, lat[[i1, rt]] is the lattice site to the right of the site, and lat[[i1, lt]] is the lattice site to the left of the site. The values of dn, up, rt and lt are determined by

```
If[i1 == n, dn = 1, dn = i1 + 1];
If[i2 == n, rt = 1, rt = i2 + 1];
If[i1 == 1, up = n, up = i1 - 1];
If[i2 == 1, lt = n, lt = i2 - 1];
```

```
If[condition, t, f] gives t if condition evaluates to True, and f if it evaluates to False.
```

(3b) The energy change resulting from flipping the spin of the selected lattice site is given by

```
2 lat[[i1, i2]] (B + J nnvalsum)
```

where **nnvalsum** is the total spin from the neighbors

```
nnvalsum = lat[[dn, i2]] + lat[[up, i2]] + lat[[i1, rt]] + lat[[i1, lt]]
```

```
nnvalsum = lat[[dn, i2]] + lat[[up, i2]] + lat[[i1, rt]] + lat[[i1, lt]]
```

```
-2
```

The value of **lat**[[i1, i2]] is either 1 or -1 and the value of **nnvalsum** is either 4, 2, 0, -2, or -4 (4 corresponds to all spins up, 2 to three spins up and one spin down, 0 to two spins up and two spins down, -2 to three spins down and one spin up and -4 to all spins down). Thus, there are ten possible energy changes as a result of spin flipping.

Rather than spending time calculating the energy change for flipping the selected site, we create a 'look-up' table of the possible energy changes

```
energydiff[1, 4] = 2 (B + 4 J);
energydiff[1, 2] = 2 (B + 2 J);
energydiff[1, 0] = 2 (B + 0 J);
energydiff[1, -2] = 2 (B - 2 J);
energydiff[1, -4] = 2 (B - 4 J);
energydiff[-1, 4] = -2 (B + 4 J);
energydiff[-1, 2] = -2 (B + 2 J);
energydiff[-1, 0] = -2 (B + 0 J);
energydiff[-1, -2] = -2 (B - 2 J);
energydiff[-1, -4] = -2 (B - 4 J);
```

note: By convention, a negative **energydiff** corresponds to a lowering of the lattice energy and a positive e**nergydiff** corresponds to a raising of the lattice energy.

(4) It is decided whether to flip the spin of the selected lattice site using the following criteria:

(4a) The energy of the lattice is lowered as a result of the flip (ie., if **energydiff** is negative).

```
energydiff[lat[[i1, i2]], nnvalsum] < 0
```

```
B = 0; J = 1; energydiff[lat[[i1, i2]], nnvalsum] < 0
```

```
False
```

(4b) The energy of the lattice is raised as a result of the flip, but the exponential of (-**energydiff**) is greater than a random number between 0 and 1.

```
Random[] < Exp[-energydiff[lat[[i1, i2]], nnvalsum]
```

note: The more the energy is raised by the flip, the more positive **energydiff** is. Hence, the smaller the exponential of (-**energydiff**) is and the less likely that it will be greater than Random[].

```
Random[] < Exp[-energydiff[lat[[i1, i2]], nnvalsum]]
```

```
False
```

(4c) If either of these conditions are met, the spin of the selected site is flipped using

lat[[i1, i2]] = -**lat**[[i1, i2]]

(4d) The final lattice configuration is returned

lat

The Metropolis method given in steps 4a-d is expressed using a conditional function

```
If[energydiff[lat[[i1, i2]], nnvalsum] < 0 ||
     Random[] < Exp[-energydiff[lat[[i1, i2]], nnvalsum]],
   lat[[i1, i2]] = -lat[[i1, i2]]; lat, lat]
```

```
If[energydiff[lat[[i1, i2]], nnvalsum] < 0 ||
  Random[] < Exp[-energydiff[lat[[i1, i2]], nnvalsum]],
 lat[[i1, i2]] = -lat[[i1, i2]]; lat, lat]; MatrixForm[lat]
```

$$\begin{pmatrix} 1 & 1 & -1 & 1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 \end{pmatrix}$$

We can combine steps 2-4 in an anonymous function, using the symbol # to represent the lattice configuration.

```
flip = (lat = #1; {i1, i2} = {Random[Integer, {1, n}], Random[Integer, {1, n}]};
   If[i1 == n, dn = 1, dn = i1 + 1]; If[i2 == n, rt = 1, rt = i2 + 1];
   If[i1 == 1, up = n, up = i1 - 1]; If[i2 == 1, lt = n, lt = i2 - 1];
   nnvalsum = lat[[dn, i2]] + lat[[up, i2]] + lat[[i1, rt]] + lat[[i1, lt]];
   If[energydiff[lat[[i1, i2]], nnvalsum] < 0 || Random[] < e^-energydiff[lat[[i1,i2]],nnvalsum],
    lat[[i1, i2]] = -lat[[i1, i2]]; lat, lat]) &
```

Consultem a ajuda sobre a definicao de funcoes: *body & is a pure function with arguments specified as # or #1, #2, etc.*

(5) The repeated application of the sequence of steps 2-4, m times, building a list of the lattice configuration after each flip attempt, is performed using the NesList function with the starting lattice configuration and the flip function

```
flipLis = NestList[flip, lat, m]
```

Um passo de Monte Carlo corresponde a n^2 operações - um ensaio de flip de spin por site da rede. Portanto, o número de vezes que queremos executar a operação é o produto do número de passos de MC, mcs, pelo tamanho da rede, n^2.

```
mcs = 2; m = mcs * n^2; flipLis = NestList[flip, lat, m]
```

```
{{{1, 1, -1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, -1}},
 {{1, 1, -1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, -1}},
 {{1, 1, -1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, -1}},
 {{1, 1, -1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, -1}},
 {{1, 1, -1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{1, -1, -1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{1, -1, -1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{1, -1, -1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{1, -1, -1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{1, -1, -1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{1, 1, -1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{1, 1, 1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{1, 1, 1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{1, 1, 1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{1, 1, 1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{1, 1, 1, 1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{1, 1, 1, 1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{1, -1, 1, 1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{1, -1, 1, 1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, -1}},
 {{1, -1, 1, 1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, -1}},
 {{1, -1, 1, 1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{-1, -1, 1, 1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{-1, -1, 1, -1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{-1, -1, 1, -1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{-1, -1, 1, -1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{-1, -1, 1, -1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{-1, -1, 1, -1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, -1}},
 {{-1, -1, 1, -1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, -1}},
 {{-1, -1, 1, -1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, -1}},
 {{-1, -1, 1, -1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, -1}},
 {{-1, -1, 1, -1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, -1}}}
```

(6) A sublist, **monteCarloStepLis**, containing every (n^2)th element from the list of lattice configurations is created using

```
monteCarloStepLis = flipLis[[Range[1, m, n^2]]]
```

`Range[`*imin*`, `*imax*`]` generates the list {*imin*, … , *imax*}. `Range[`*imin*`, `*imax*`, `*di*`]` uses step *di*.

```
Range[1, m + 1, n^2]
```

```
{1, 17, 33}
```

Uso esta lista para guardar a configuração do sistema ao fim de cada passo de MC, ou seja, ao fim de cada conjunto de n^2 iterações.

```
monteCarloStepLis = flipLis[[Range[1, m + 1, n^2]]]
```

```
{{{1, 1, -1, 1}, {-1, 1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, -1}},
 {{1, 1, 1, 1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, 1}},
 {{-1, -1, 1, -1}, {-1, -1, -1, -1}, {-1, -1, -1, -1}, {1, -1, -1, -1}}}
```

(7) The long range order, which is the absolute value of the magnetization of the lattice, is determined for each lattice configuration in monteCarloStepLis using

```
(Abs[Plus @@ Flatten[#1] / n^2] &) /@ monteCarloStepLis
```

```
(Abs[Plus @@ Flatten[#1] / n^2] &) /@ monteCarloStepLis
```

$$\left\{\frac{3}{8}, \frac{1}{4}, \frac{3}{4}\right\}$$

We can construct the program for the Ising model from these code fragments.

The Program

```
Null
```

```
isingMetropolis[n_, m_, B_, J_] :=
 Module[{energydiff, initconfig, flip, flipLis, monteCarloStepLis},
  energydiff[1, 4] = 2 (B + 4 J); energydiff[1, 2] = 2 (B + 2 J);
  energydiff[1, 0] = 2 (B + 0 J); energydiff[1, -2] = 2 (B - 2 J);
  energydiff[1, -4] = 2 (B - 4 J); energydiff[-1, 4] = -2 (B + 4 J);
  energydiff[-1, 2] = -2 (B + 2 J); energydiff[-1, 0] = -2 (B + 0 J);
  energydiff[-1, -2] = -2 (B - 2 J); energydiff[-1, -4] = -2 (B - 4 J);
  initconfig = 2 Table[Random[Integer], {n}, {n}] - 1;
  flip = (lat = #1; {i1, i2} = {Random[Integer, {1, n}], Random[Integer, {1, n}]};
     If[i1 == n, dn = 1, dn = i1 + 1]; If[i2 == n, rt = 1, rt = i2 + 1];
     If[i1 == 1, up = n, up = i1 - 1]; If[i2 == 1, lt = n, lt = i2 - 1];
     nnvalsum = lat[[dn, i2]] + lat[[up, i2]] + lat[[i1, rt]] + lat[[i1, lt]];
     If[energydiff[lat[[i1, i2]], nnvalsum] < 0 || Random[] < e^-energydiff[lat[[i1,i2]],nnvalsum],
      lat[[i1, i2]] = -lat[[i1, i2]]; lat, lat]) &; flipLis =
   NestList[flip, initconfig, m]; monteCarloStepLis = flipLis[[Range[1, m + 1, n^2]]]
```

```
isingMetropolis[4, 2 * 16, 0, 1]
```

```
{{{-1, 1, 1, -1}, {1, 1, -1, 1}, {1, -1, 1, 1}, {-1, 1, 1, -1}},
 {{-1, 1, 1, 1}, {-1, 1, -1, 1}, {-1, -1, -1, 1}, {-1, 1, 1, 1}},
 {{-1, 1, 1, 1}, {-1, -1, 1, 1}, {-1, -1, 1, 1}, {-1, -1, 1, 1}}}
```

Magnetization Behavior of the Ising Model

The magnetization behavior over time can be computed using the program

```
ShowIsingListPlot[list_] :=
 Module[{n = Length[list]}, longRangeOrderList = (Abs[Plus @@ Flatten[#1] / (40 * 40)] &) /@ list;
  ListPlot[longRangeOrderList, PlotJoined → True, PlotRange → All,
   PlotLabel → FontForm["Long-range order parameter", {"Times", 12}]];]
```

Running the Program

```
ShowIsingListPlot[isingMetropolis[40, 40 * 40 * 100, 0., 1.0]]
```

```
No more memory available.
Mathematica kernel has shut down.
Try quitting other applications and then retry.
```

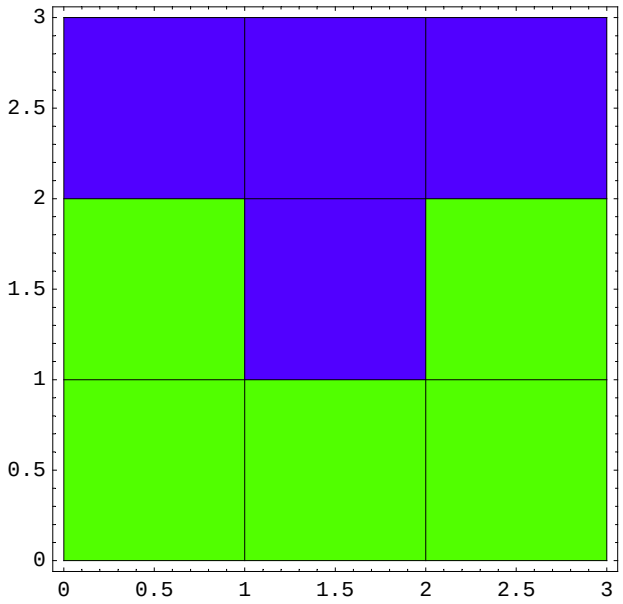
Vamos então ensaiar o comportamento do sistema para uma rede mais pequena, e para isso temos que adaptar a normalização da magnetização global na definição do comando anterior.

```
ShowIsingListPlot[list_] :=
 Module[{n = Length[list]}, longRangeOrderList = (Plus @@ Flatten[#1] / (15 * 15) &) /@ list;
  ListPlot[longRangeOrderList, PlotJoined → True, PlotRange → All,
   PlotLabel → FontForm["Long-range order parameter", {"Times", 12}]];]
```

Para representar graficamente as configurações do sistema é conveninete definir uma outra função.

```
ShowIsing[list_, opts___] :=
 ListDensityPlot[list, ColorFunction :> (If[#1 == 0, Hue[.72], Hue[.28]] &), opts]
```

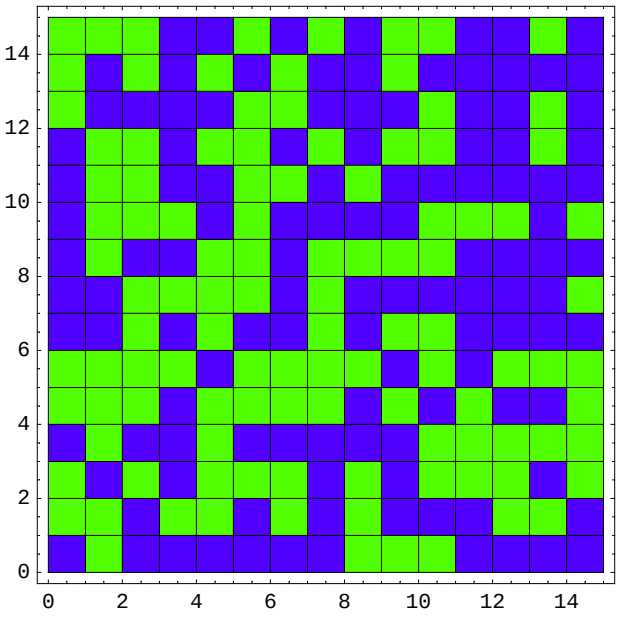
```
test = {{1, 1, 1}, {1, -1, 1}, {-1, -1, -1}}; ShowIsing[test]
```




```
- DensityGraphics -
```

A seguir usa-se esta função para representar a configuração inicial dos cpins do sistema:

```
ShowIsing[isingMetropolis[15, 15 * 15 * 2, 0., 0.43 - 0.2 + 4 * 0.05][[1]]]
```




```
- DensityGraphics -
```