

# Monte-Carlo Study of the 2D Ising Model

Michael Papasimeon

August 25, 1997

**Abstract**—This paper was written for an introductory undergraduate class in computational physics in 1997. It focuses on Monte-Carlo simulation of the 2D Ising model. The work models a regular grid of particles each with spin up or down. The work attempts to determine the critical temperature  $\beta_c$  where a phase transition occurs. The source code listings in FORTRAN77 can be found in the appendices.

## I. AIMS

The main aims are to:

- Write a FORTRAN program which can represent a 2D lattice of spin up/down particles using the 2D Ising model.
- Extend the program to use a Monte-Carlo technique to investigate the magnetisation of the lattice for different temperatures.
- Use the program to attempt to determine the critical temperature (where a phase transition occurs) for the ferro-magnetic Ising model.

## II. THEORY

Figure 1 represents a two dimensional lattice of  $N_s = N_x \times N_y$  particles. Each particle can be either spin up or spin down. The spin at site  $\alpha = (i, j)$  is given by  $\sigma_\alpha^z$ . A lattice of  $N_s$  particles can be in  $2^{N_s}$  possible configurations, represented as  $\underline{\sigma}^z = (\sigma_1^z, \dots, \sigma_{N_s}^z)$ . The total energy for a given configuration is given by:

$$E(\underline{\sigma}^z) = -J \sum_{\langle \alpha \beta \rangle} \sigma_\alpha^z \sigma_\beta^z - B_z \sum_{\alpha} \sigma_\alpha^z$$

For this experiment we take  $B_z = 0$  because there is no external magnetic field and  $J = 1$ , because we are considering the case of a ferromagnet ( $J > 0$ ), to get:

$$E(\underline{\sigma}^z) = - \sum_{\langle \alpha \beta \rangle} \sigma_\alpha^z \sigma_\beta^z$$

We can determine the partition function of the system because we know from statistical mechanics that each configuration is a micro-state of the canonical ensemble. The partition function is then given by

$$Z = \sum_{\underline{\sigma}^z} e^{-\beta E(\underline{\sigma}^z)}$$

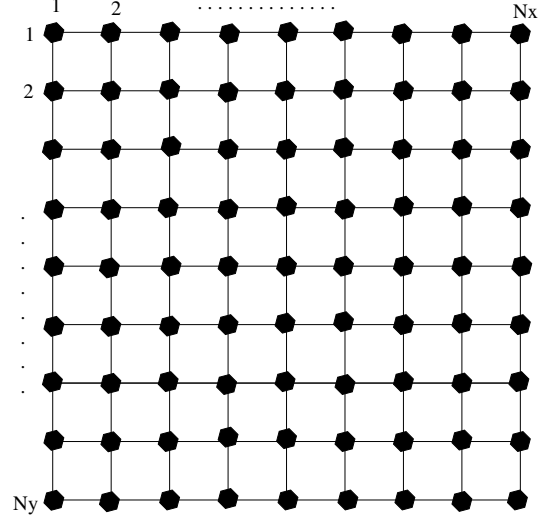
. The probability of obtaining a particular state is given by

$$P(\underline{\sigma}^z) = \frac{e^{-\beta E(\underline{\sigma}^z)}}{Z}$$

The ensemble average of a physical observable property is:

$$X_{obs} = \langle X \rangle = \sum_{\underline{\sigma}^z} X(\underline{\sigma}^z) P(\underline{\sigma}^z)$$

Fig. 1. 2D lattice of  $N_x \times N_y$  particles.



Therefore the magnetisation of the entire lattice is given by summing all the spins at all lattice sites.

$$M(\underline{\sigma}^z) = \sum_{\alpha=1}^{N_s} \sigma_\alpha^z$$

We can obtain a number of configurations and determine their magnetisation using the Metropolis algorithm. This involves sweeping through the lattice and selecting a number of sites at random. The magnetisation can then be calculated to be approximated to be the Monte-Carlo average given by:

$$M = \langle M \rangle \approx \frac{1}{N} \sum_{k=1}^N M_k$$

The weight function used in the Monte-Carlo algorithm is the Boltzmann distribution. At the randomly selected sites  $(i, j)$

```

flip_spin(i,j)
if (change in energy < 0)
    accept
else
    if (exp(-beta * change in energy) > eta)
        accept
    else
        reject

```

The energy of an entire lattice  $E$  is the sum of the energies of all the particles in the lattice. The energy of a single particle is has contributions from it's nearest neighbours in the following

way:

$$\epsilon = -J\sigma_{i,j} [\sigma_{i-1,j} + \sigma_{i,j+1} + \sigma_{i+1,j} + \sigma_{i,j-1}]$$

### III. METHOD

The following method was followed in this experiment:

- A FORTRAN program (ising.f in Appendix A), was written which represented a 2D lattice of spin up/down particles.
  - The user of the program has the option of starting the lattice with a cold start (all spins up) or a hot start with all spins randomly selected to be up or down using the pseudo-random number generating function rand()
  - A function to print the lattice on the screen was written
- A function to calculate the energy of a lattice was written making use of periodic boundary conditions.
- A function to determine the the magnetisation of a particular lattice configuration was written.
- A function to apply the Metropolis algorithm to a given lattice,  $\beta$  and for user specified number of sweeps. The function allows 100 sweeps to allow the lattice to thermalise. The magnetisation is and standard deviation are calculated over 100 configurations, with each configuration separated by 10 sweeps.
- The Metropolis function is called a number of times for different values of  $\beta$  ranging from 0.1 to 1.0.

### IV. RESULTS

Results from the simulation are shown in Figures 2 to 15. The figures plot the absolute value of the magnetisation  $|M|$  against the inverse of the temperature  $\beta$ .

The program was run for a number of different lattices sizes. The lattices sizes used are:  $4 \times 4$ ,  $6 \times 6$ ,  $8 \times 8$ ,  $10 \times 10$ ,  $12 \times 12$ ,  $14 \times 14$  and  $16 \times 16$ .

There are two sets of plots. Set 1 (Figures 2 to 8) contains plots of the magnetisation against the inverse temperature over 100 configurations and 10 sweeps per configuration. Set 2 (Figures 9 to 15) contains plots over 50 configurations with 40 sweeps per configuration.

In all the plots the error bars show the error in the magnetisation, where we have:

$$M = \langle M \rangle \pm \sigma_M$$

The error bars are the standard deviation in  $M$ ,  $\sigma_M$ . The variance  $\sigma_M^2$  is given by:

$$\sigma_M^2 = \frac{1}{N} (\langle M^2 \rangle - \langle M \rangle^2)$$

Using the graphs, we can determine the temperature at which the ferromagnetic phase transition occurs, the critical temperature  $\beta_c$ . Table I summarises these results.

TABLE I  
APPROXIMATE ESTIMATES FOR CRITICAL TEMPERATURE  $\beta_c$  FROM  
MAGNETISATION PLOTS.

$N_x \times N_y$	$N_s$	Approx $\beta_c$ (Set 1)	Approx $\beta_c$ (Set 2)
$4 \times 4$	16	0.2	0.18
$6 \times 6$	36	0.2	0.2
$8 \times 8$	64	0.2	0.2
$10 \times 10$	100	0.2	0.2
$12 \times 12$	144	0.2	0.15
$14 \times 14$	196	0.21	0.2
$16 \times 16$	256	0.3	0.2

### V. DISCUSSION

In all the simulations of different sized lattices, it is clear to see in the plots of the magnetisation that there is a phase transition occurring.

What is not clear however is the exact critical temperature  $\beta_c$ . In most of the cases (for both sweep/configuration setups) the critical temperature is approximately  $\beta_c \approx 0.2$ . In the large lattice limit we expect to get a value for the critical temperature of  $\beta_c = 0.4407$ . The only result which approaches this is that of Set 1 for a  $16 \times 16$  lattice. For the case of the  $8 \times 8$  lattice we obtain  $\beta_c \approx 0.2$ . Without trying larger lattice sizes we are unable to tell if this is realistic or merely a statistical aberration.

The magnetisation in the temperature ranging from  $\beta = 0.1$  to the critical temperature  $\beta_c$ , tends to be quite erratic in all the simulations, with large errors in a number of cases. This made determining the critical temperature difficult, and therefore only approximate results could be obtained.

The question of finding a relationship between the total lattice size  $N_s$  and the critical temperature  $\beta_c$  cannot be answered because the majority of the results indicate that  $\beta_c \approx 0.2$  for  $16 \leq N_s \leq 256$ . Due to the statistical nature of this simulation, it would be more realistic to go through many runs for each lattice size and then take the mean of the estimated critical temperatures.

### VI. PLOTS OF MAGNETISATION AS A FUNCTION OF INVERSE TEMPERATURE

This appendix contains the results from all the simulations conducted. The figures show inverse temperature  $\beta$  on the  $x$ -axis and the grid magnetisation  $|M|$  on the  $y$ -axis. The figures show simulations of various grid sizes ranging from a small  $4 \times 4$  grid all the way up to a  $16 \times 16$  grid. The results are divided into two sets.

Set 1 (Figures 2 to 8) contains plots of the magnetisation against the inverse temperature over 100 configurations and 10 sweeps per configuration.

Set 2 (Figures 9 to 15) contains plots over 50 configurations with 40 sweeps per configuration.

A. Set 1: 100 configurations with 10 sweeps/configuration

B. Set 2: 50 Configurations and 40 Sweeps/Configuration

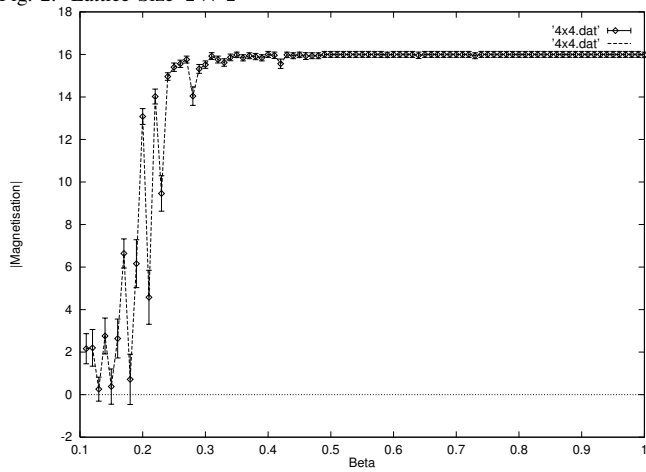
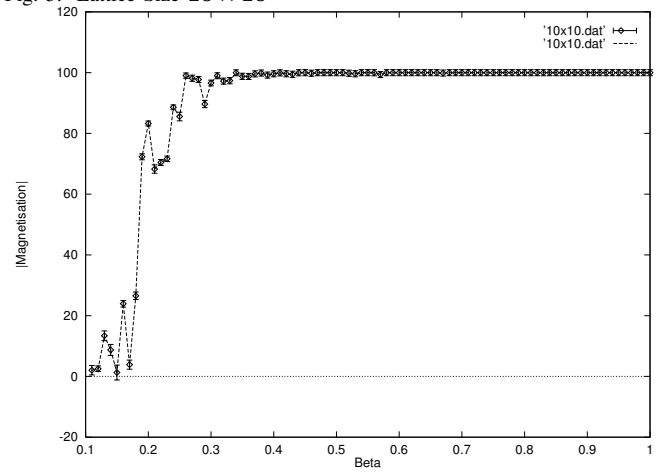
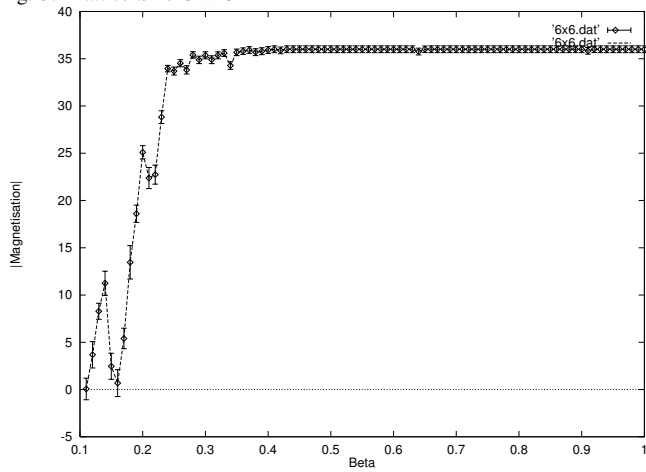
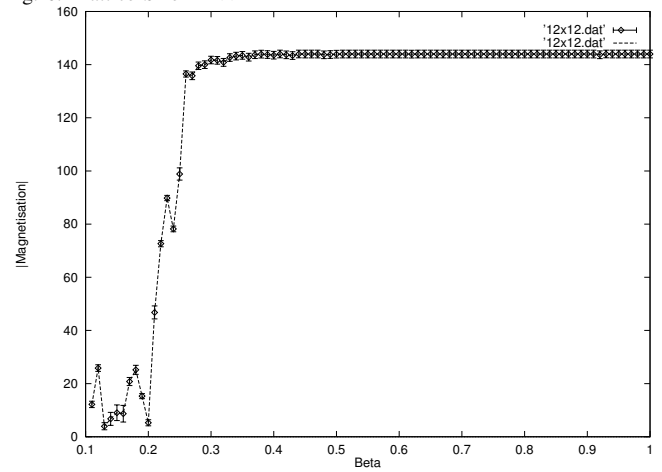
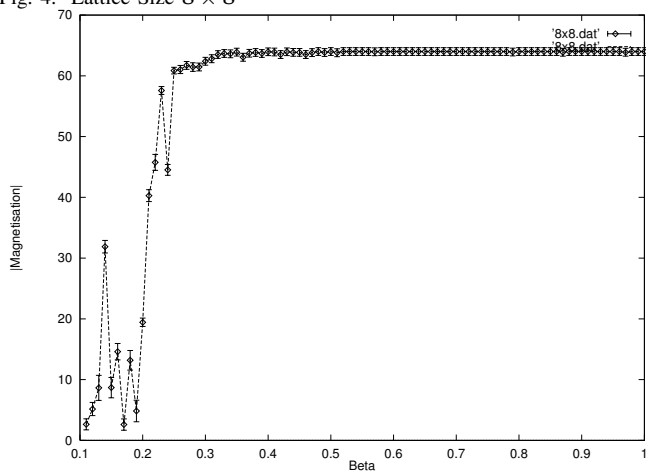
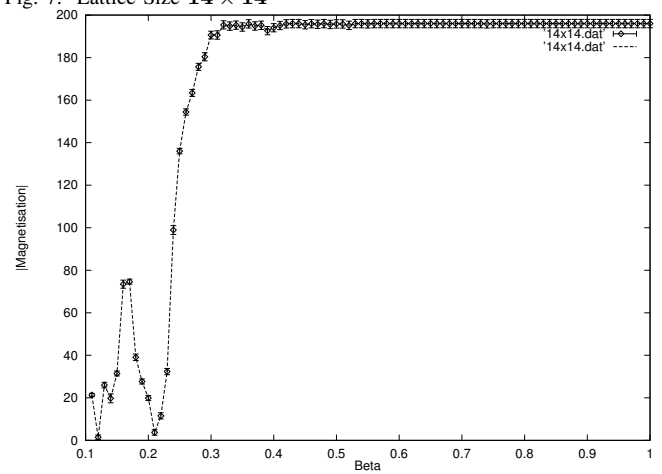
Fig. 2. Lattice Size  $4 \times 4$ Fig. 5. Lattice Size  $10 \times 10$ Fig. 3. Lattice Size  $6 \times 6$ Fig. 6. Lattice Size  $12 \times 12$ Fig. 4. Lattice Size  $8 \times 8$ Fig. 7. Lattice Size  $14 \times 14$ 

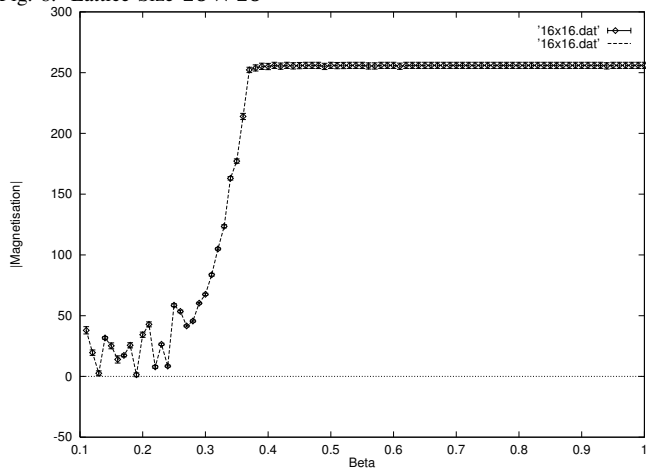
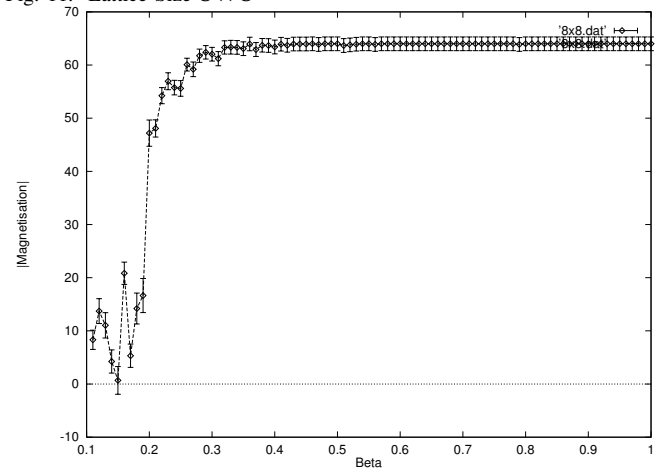
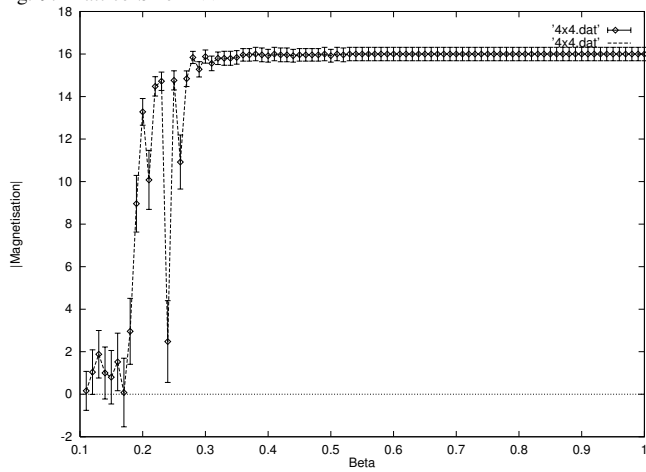
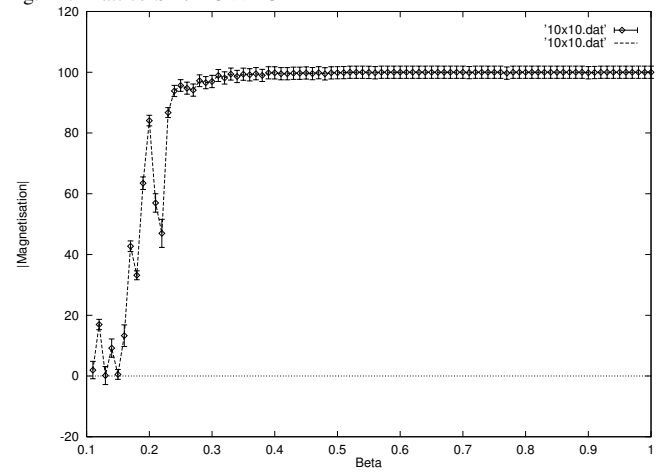
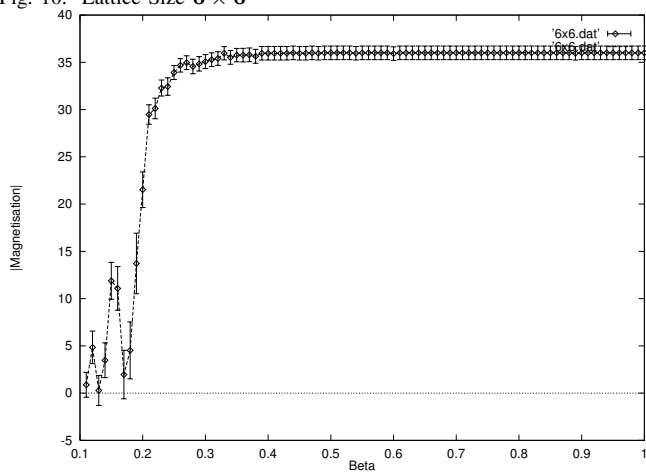
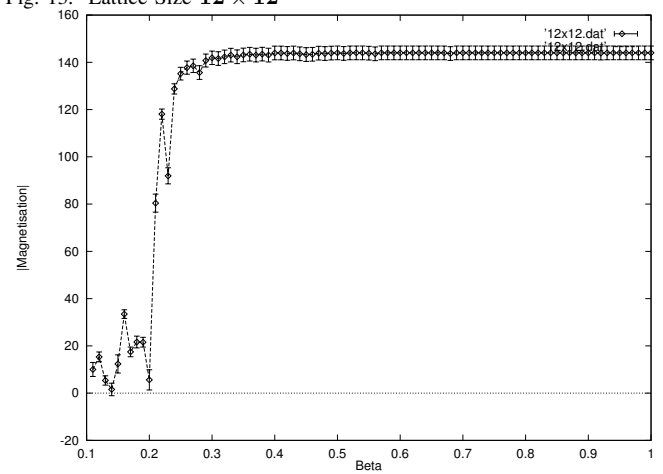
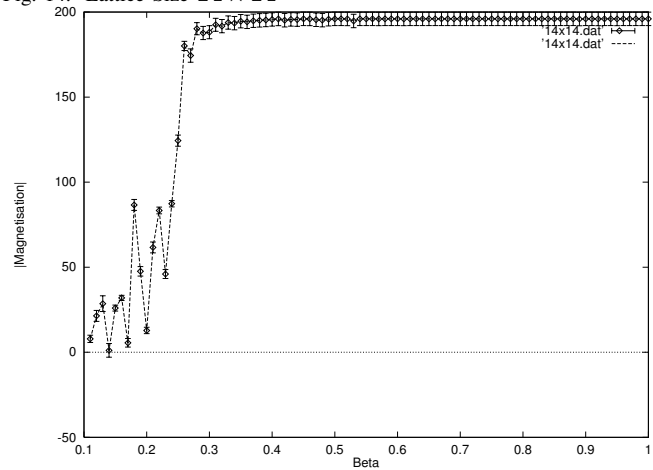
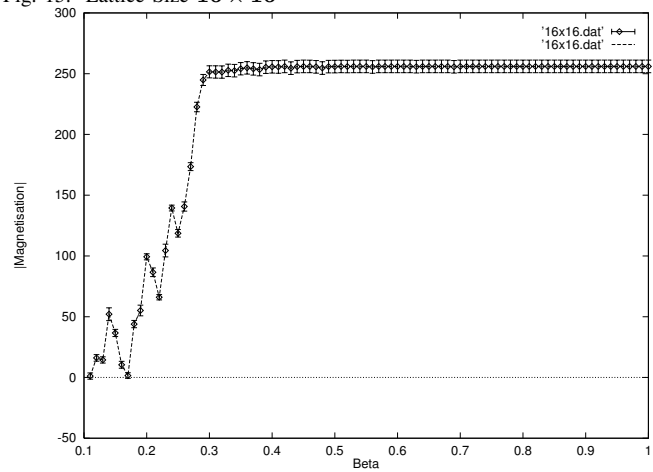
Fig. 8. Lattice Size  $16 \times 16$ Fig. 11. Lattice Size  $8 \times 8$ Fig. 9. Lattice Size  $4 \times 4$ Fig. 12. Lattice Size  $10 \times 10$ Fig. 10. Lattice Size  $6 \times 6$ Fig. 13. Lattice Size  $12 \times 12$ 

Fig. 14. Lattice Size  $14 \times 14$ Fig. 15. Lattice Size  $16 \times 16$ 

APPENDIX  
CODE LISTING: ISING.F

```

1  C-----
2
3      program main
4          implicit none
5          integer seed, Nx, Ny, i, j, k
6              integer start, generate_spin, energy
7          parameter(Nx=16)
8          parameter(Ny=16)
9          integer spins(Nx,Ny)
10             integer ene, Ns
11             real*8 beta
12
13             open(unit=1, file='16x16.dat', status='unknown')
14
15             write(*,*)'Enter_Seed_for_Random_Number_Generator'
16             read(*,*) seed
17             call srand(seed)
18
19             write(*,*)'Enter_1_for_a_Cold_Start_and_2_for_a_Hot_Start'
20             read(*,*) start
21
22             do i = 1, Nx, +1
23                 do j = 1, Ny, +1
24                     spins(i,j) = generate_spin(start)
25                 end do
26             end do
27
28  c call print_lattice(spins, Nx, Ny)
29
30             ene = energy(spins, Nx, Ny)
31             write(*,*)'Initial_Energy= ',ene
32
33             Ns = 1000
34             beta = 0.1d0
35             do k = 1, 90, +1
36                 beta = beta + 0.01d0
37                 call metropolis(spins, beta, Ns, Nx, Ny)
38             end do
39
40             close(unit=1)
41         end
42
43  C-----
44
45      integer function generate_spin(start)
46          implicit none
47          integer start, s
48          real*8 rval, rand
49
50          if (start .eq. 1) then
51              s = 1
52          else
53              rval = rand()
54              if (rval .lt. 0.5d0) then
55                  s = -1
56              else
57                  s = 1
58              end if
59          end if
60
61          generate_spin = s
62          return
63      end
64

```

```

65 C-----
66
67     subroutine print_lattice(spins, Nx, Ny)
68     implicit none
69     integer i, j, Nx, Ny
70     integer spins(Nx,Ny)
71
72     do i = 1, Nx, +1
73         do j = 1, Ny, +1
74             if (spins(i,j) .eq. 1) then
75                 write(*,100) '+'
76             end if
77             if (spins(i,j) .eq. -1) then
78                 write(*,100) '-'
79             end if
80 100 format(a2, $)
81         end do
82         write(*,*)
83     end do
84     end
85
86 C-----
87
88     integer function energy(spins,Nx,Ny)
89     implicit none
90     integer i, j, Nx, Ny
91     integer spins(Nx,Ny)
92     integer left, right, up, down
93     integer en, total_energy
94
95     en = 0
96     total_energy = 0
97     left = 1
98     right = 1
99     up = 1
100    down = 1
101
102    do i = 1, Nx, +1
103        do j = 1, Ny, +1
104
105            left = i - 1
106            right = i + 1
107            up = j - 1
108            down = j + 1
109
110            if (i .eq. 1) then
111                left = Nx
112            end if
113
114            if (i .eq. Nx) then
115                right = 1
116            end if
117
118            if (j .eq. 1) then
119                up = Ny
120            end if
121
122            if (j .eq. Ny) then
123                down = 1
124            end if
125
126            en = -spins(i,j)*(spins(left,j)+
127 > spins(i,up)+spins(right,j)+spins(i,down))
128            total_energy = total_energy + en
129        end do
130    end do
131
132    energy = total_energy

```

```

133         return
134     end
135
136 C-----
137
138     subroutine metropolis(spins, beta, Ns, Nx, Ny)
139         implicit none
140         real*8 M, beta, eta, rand, error
141         integer i_t, j_t, energy, currentEnergy, newEnergy
142         integer magnetisation
143         integer deltaEnergy
144         integer sweep, i, j, Nx, Ny, Ns
145         integer spins(Nx,Ny)
146         integer thermalise, N, mag, msquared
147
148         thermalise = 100
149         N = 0
150         M = 0
151
152         do sweep = 1, (Ns + thermalise), +1
153             currentEnergy = energy(spins, Nx, Ny)
154
155             i_t = idint(Nx*rand()) + 1
156             j_t = idint(Ny*rand()) + 1
157             spins(i_t, j_t) = -spins(i_t, j_t)
158
159             newEnergy = energy(spins, Nx, Ny)
160             deltaEnergy = newEnergy - currentEnergy
161             eta = rand()
162
163             if (deltaEnergy .ge. 0) then
164                 if (dexp(-beta*deltaEnergy) .le. eta) then
165                     spins(i_t, j_t) = -spins(i_t, j_t)
166                 end if
167             end if
168
169 C     call print_lattice(spins, Nx, Ny)
170 C     pause
171
172             if (sweep .gt. thermalise) then
173                 if (mod(sweep, 10) .eq. 0d0) then
174                     N = N + 1
175                     mag = magnetisation(spins, Nx, Ny)
176                     M = M + mag
177                     msquared = msquared + mag**2
178                 end if
179             end if
180         end do
181
182         M = M/dfloat(N)
183         msquared = msquared/dfloat(N)
184         error = dsqrt((msquared - M**2)/dfloat(N))
185         write(*,*) 'beta, M, error = ', beta, dabs(M), error
186         write(1,*) beta, dabs(M), error
187     end
188 C-----
189
190     integer function magnetisation(spins, Nx, Ny)
191         implicit none
192         integer i, j, Nx, Ny
193         integer spins(Nx,Ny)
194         integer mag
195
196         mag = 0
197
198         do i = 1, Nx, +1
199             do j = 1, Ny, +1
200                 mag = mag + spins(i,j)

```



```
201         end do
202     end do
203
204     magnetisation = mag
205     return
206 end
```