Monte-Carlo Study of the 2D Ising Model

Michael Papasimeon
August 25, 1997

***Abstract*—**

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 σ^z_α. A lattice of N_s particles can be in 2^{N_s} possible configurations, represented as $\underline{\sigma}^z = (\sigma^z_1, ..., \sigma^z_{N_s})$. The total energy for a given configuration is given by:

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

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

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

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

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

. The probability of obtaining a particular state is given by

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

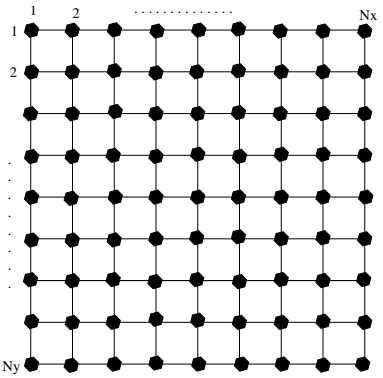
The ensemble average of a physical observable property is:

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

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

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

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



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:

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

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, β 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 β ranging from 0.1 to 1.0.

IV. Results

The plots below in this section absolute values of the magnetisation $|M|$ against the inverse of the temperature β.

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

There are two sets of plots. Set 1 contains plots of the magnetisation against the inverse temperature over 100 configurations and 10 sweeps per configuration. Set 2 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, σ_M. The variance σ_M^2 is given by:

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

Using the graphs, we can determine the temperature at which the ferromagnetic phase transition occurs, the critical temperature β_c. The table below summarises these results.

$N_x \times N_y$	N_s	Approx β_c (Set 1)	Approx β_c (Set 2)
4x4	16	0.2	0.18
6x6	36	0.2	0.2
8x8	64	0.2	0.2
10x10	100	0.2	0.2
12x12	144	0.2	0.15
14x14	196	0.21	0.2
16x16	256	0.3	0.2

Table 1 : Approximate Estimates for Critical Temperature β_c from Magnetisation plots.

A. Set 1: 100 Configurations and 10 Sweeps/Configuration

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

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.

Fig. 2. Lattice Size $\mathbf{4 \times 4}$

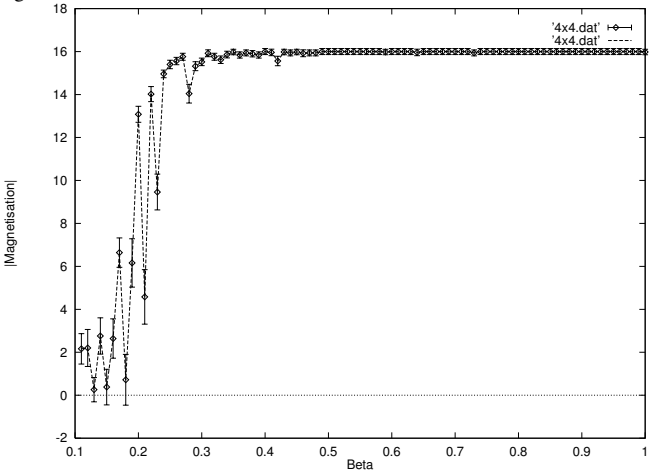


Fig. 3. Lattice Size $\mathbf{6 \times 6}$

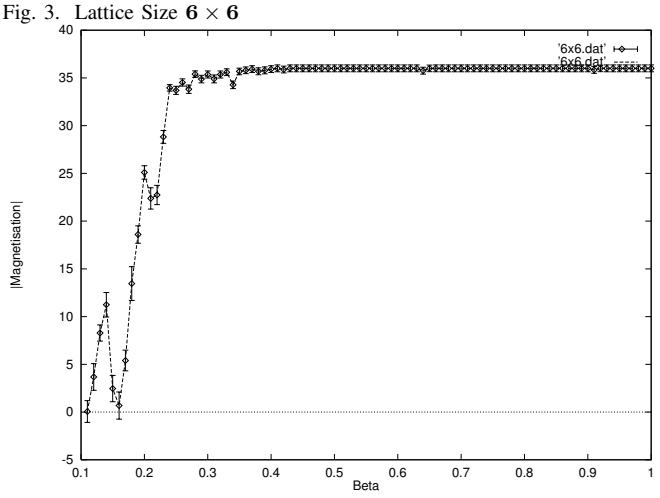


Fig. 4. Lattice Size $\mathbf{8 \times 8}$

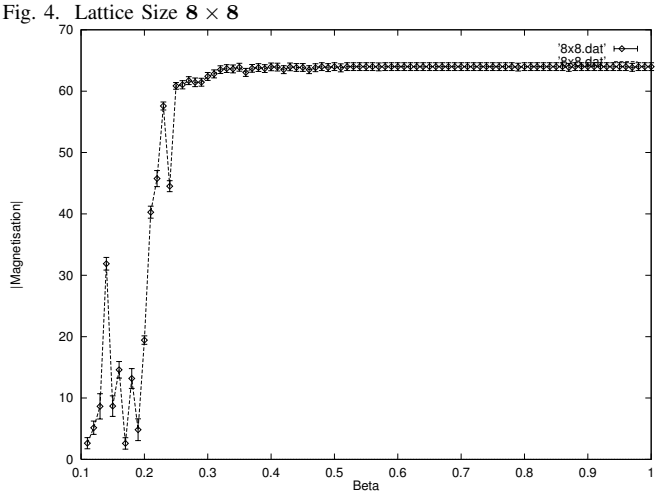


Fig. 5. Lattice Size $\mathbf{10 \times 10}$

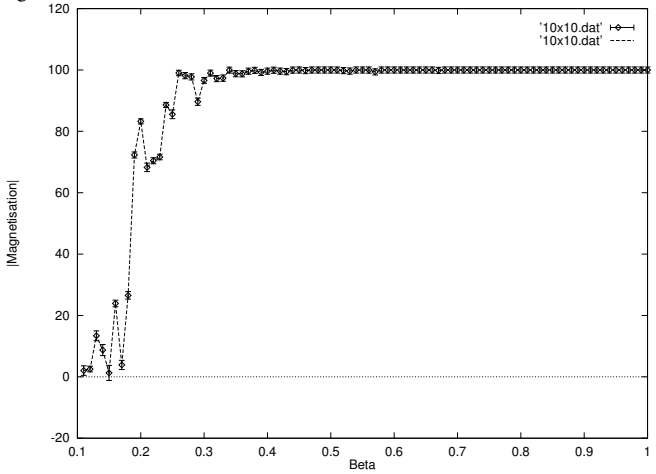


Fig. 6. Lattice Size $\mathbf{12 \times 12}$

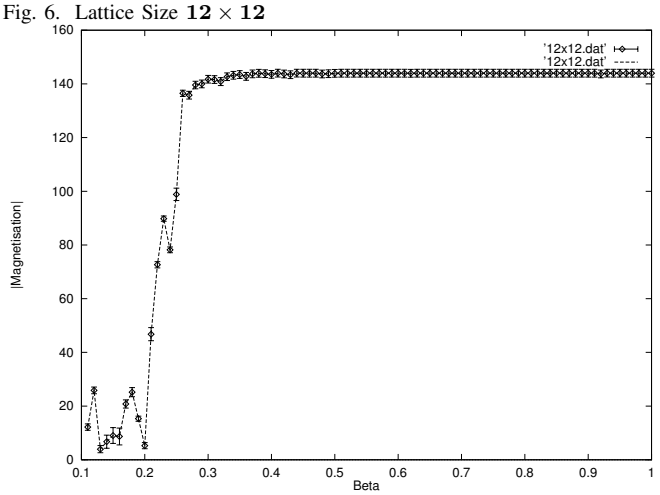


Fig. 7. Lattice Size $\mathbf{14 \times 14}$

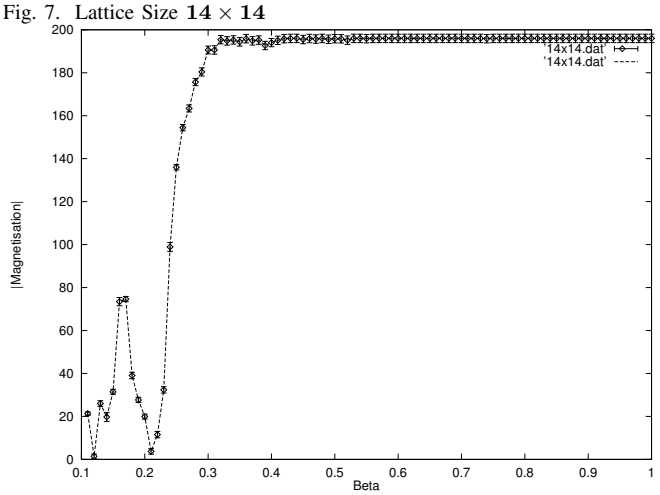


Fig. 8. Lattice Size $\mathbf{16 \times 16}$

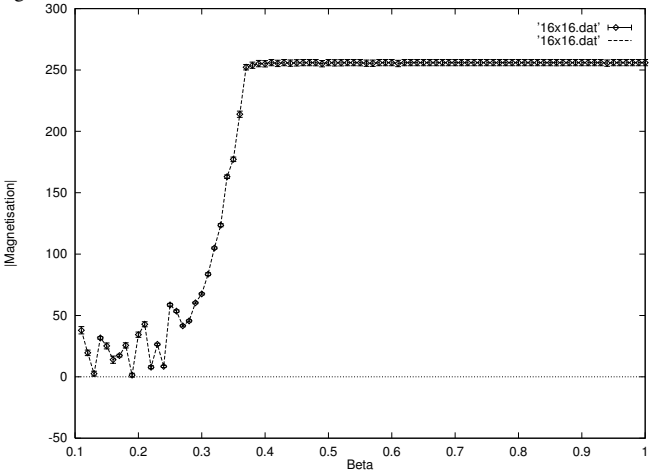


Fig. 9. Lattice Size $\mathbf{4 \times 4}$

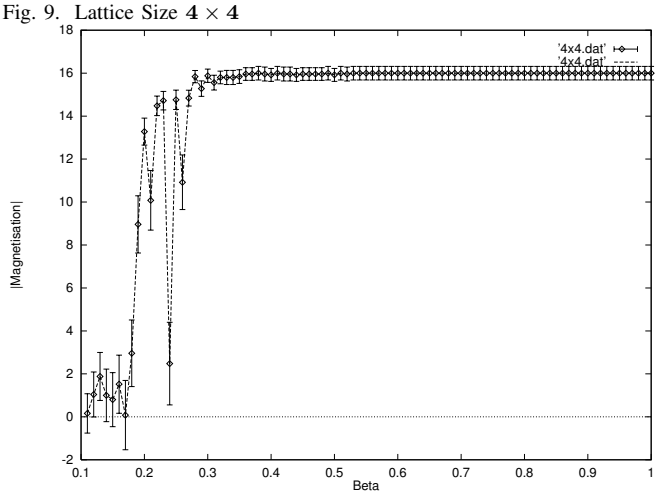


Fig. 10. Lattice Size $\mathbf{6 \times 6}$

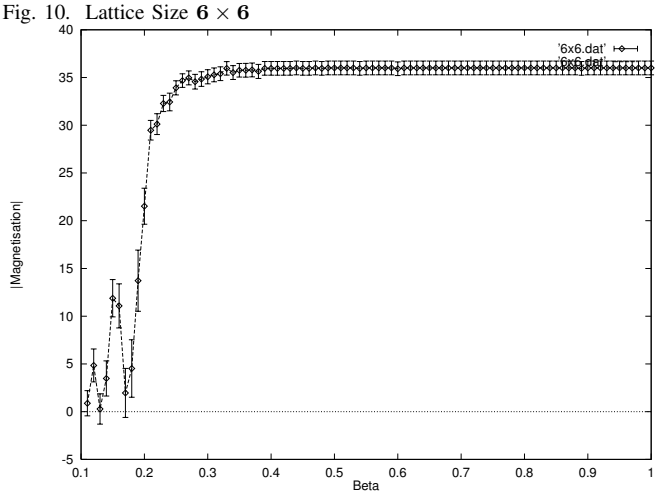


Fig. 11. Lattice Size $\mathbf{8 \times 8}$

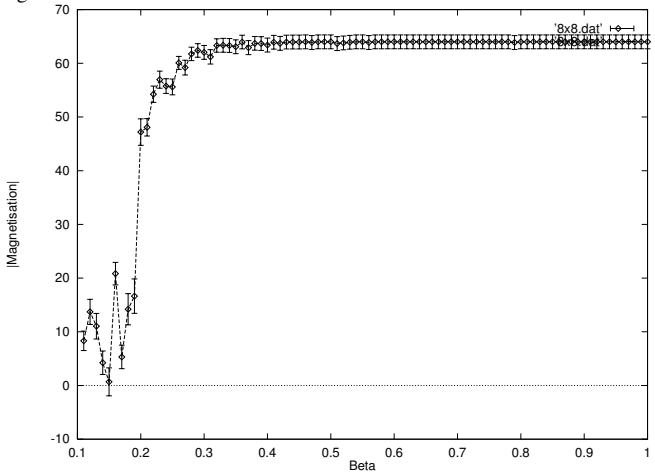


Fig. 12. Lattice Size $\mathbf{10 \times 10}$

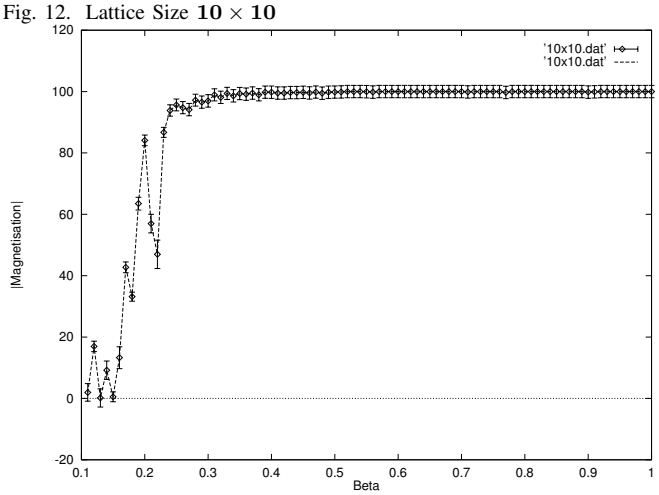


Fig. 13. Lattice Size $\mathbf{12 \times 12}$

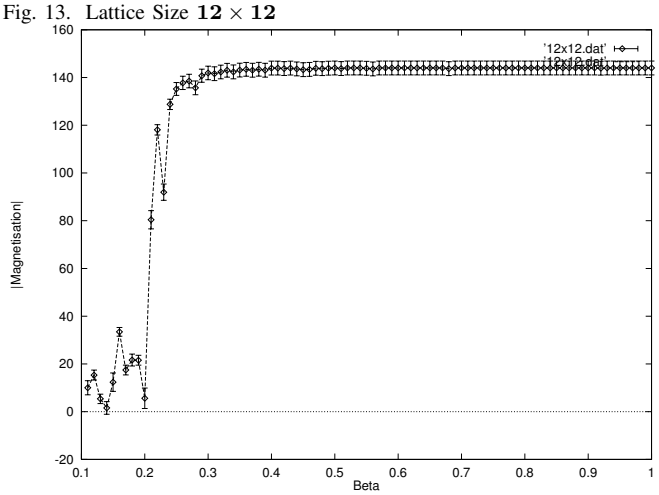


Fig. 14. Lattice Size $\mathbf{14 \times 14}$

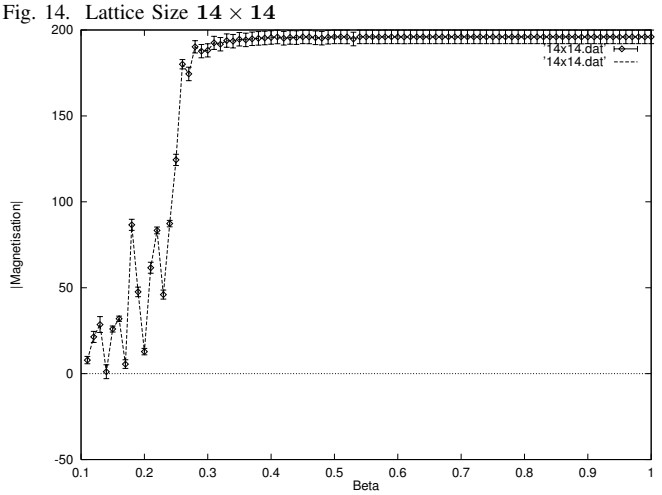
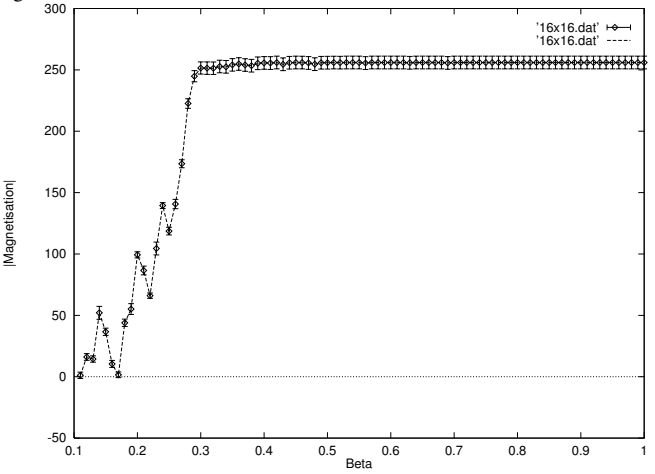


Fig. 15. Lattice Size $\mathbf{16 \times 16}$



What is not clear however is the exact critical temperature β_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 16×16 lattice. For the case of the 8×8 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 β_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 β_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.

APPENDIX
CODE LISTING: ISING.F

```
c-------------------------------------------------------------------------------

      program main
            implicit none
            integer seed, Nx, Ny, i, j, k
                integer start, generate_spin, energy
            parameter(Nx=16)
            parameter(Ny=16)
            integer spins(Nx,Ny)
                integer ene, Ns
                real*8 beta

                open(unit=1, file='16x16.dat', status='unknown')

                write(*,*)'Enter Seed for Random Number Generator'
                read(*,*), seed
                call srand(seed)

            write(*,*)'Enter 1 for a Cold Start and 2 for a Hot Start'
            read(*,*) start

            do i = 1, Nx, +1
                  do j = 1, Ny, +1
                        spins(i,j) = generate_spin(start)
                  end do
            end do

c call print_lattice(spins, Nx, Ny)

                ene = energy(spins, Nx, Ny)
                write(*,*)'Initial Energy = ',ene

                Ns = 1000
                beta = 0.1d0
                do k = 1, 90, +1
                       beta = beta + 0.01d0
                       call metropolis(spins, beta, Ns, Nx, Ny)
                end do

                close(unit=1)
      end

c-------------------------------------------------------------------------------

      integer function generate_spin(start)
            implicit none
            integer start, s
            real*8 rval, rand

            if (start .eq. 1) then
                  s = 1
            else
                  rval = rand()
                  if (rval .lt. 0.5d0) then
                        s = -1
                  else
                        s = 1
                  end if
            end if

            generate_spin = s
            return
      end

```

```
c-------------------------------------------------------------------------------

      subroutine print_lattice(spins, Nx, Ny)
            implicit none
            integer i, j, Nx, Ny
            integer spins(Nx,Ny)

            do i = 1, Nx, +1
                  do j = 1, Ny, +1
                        if (spins(i,j) .eq. 1) then
                              write(*,100) '+'
                        end if
                        if (spins(i,j) .eq. -1) then
                              write(*,100) '-'
                        end if
100 format(a2, $)
                  end do
                  write(*,*)
            end do
      end

c-------------------------------------------------------------------------------

       integer function energy(spins,Nx,Ny)
              implicit none
              integer i, j, Nx, Ny
              integer spins(Nx,Ny)
              integer left, right, up, down
              integer en, total_energy

              en = 0
              total_energy = 0
              left = 1
              right = 1
              up = 1
              down = 1

              do i = 1, Nx, +1
                     do j = 1, Ny, +1

                            left = i - 1
                            right = i + 1
                            up = j - 1
                            down = j + 1

                            if (i .eq. 1) then
                                   left = Nx
                            end if

                            if (i .eq. Nx) then
                                   right = 1
                            end if

                            if (j .eq. 1) then
                                   up = Ny
                            end if

                            if (j .eq. Ny) then
                                   down = 1
                            end if

                            en = -spins(i,j)*(spins(left,j)+
     >    spins(i,up)+spins(right,j)+spins(i,down))
                            total_energy = total_energy + en
                     end do
              end do

              energy = total_energy
```

```
                return
        end

c-------------------------------------------------------------------------------

        subroutine metropolis(spins, beta, Ns, Nx, Ny)
                implicit none
                real*8 M, beta, eta, rand, error
                integer i_t, j_t, energy, currentEnergy, newEnergy
                integer magnetisation
                integer deltaEnergy
            integer sweep, i, j, Nx, Ny, Ns
            integer spins(Nx,Ny)
                integer thermalise, N, mag, msquared

                thermalise = 100
                N = 0
                M = 0

                do sweep = 1, (Ns + thermalise), +1
                        currentEnergy = energy(spins, Nx, Ny)

                        i_t = idint(Nx*rand()) + 1
                        j_t = idint(Ny*rand()) + 1
                        spins(i_t, j_t) = -spins(i_t, j_t)

                        newEnergy = energy(spins, Nx, Ny)
                        deltaEnergy = newEnergy - currentEnergy
                        eta = rand()

                        if (deltaEnergy .ge. 0) then
                                if (dexp(-beta*deltaEnergy) .le. eta) then
                                        spins(i_t, j_t) = -spins(i_t, j_t)
                                end if
                        end if

c       call print_lattice(spins, Nx, Ny)
c       pause

                        if (sweep .gt. thermalise) then
                                if (mod(sweep, 10) .eq. 0d0) then
                                        N = N + 1
                                        mag = magnetisation(spins, Nx, Ny)
                                        M = M + mag
                                        msquared = msquared + mag**2
                                end if
                        end if
                end do

                M = M/dfloat(N)
                msquared = msquared/dfloat(N)
                error = dsqrt((msquared - M**2)/dfloat(N))
                write(*,*)'beta, M, error = ', beta, dabs(M), error
                write(1,*) beta, dabs(M), error
        end
c-------------------------------------------------------------------------------

        integer function magnetisation(spins, Nx, Ny)
                implicit none
                integer i, j, Nx, Ny
                integer spins(Nx,Ny)
                integer mag

                mag = 0

                do i = 1, Nx, +1
                        do j = 1, Ny, +1
                                mag = mag + spins(i,j)
```

```
                    end do
                end do

                magnetisation = mag
                return
        end
```