Final Projects, Part 3 & 4
The Ising Model

Finn Joel Bjervig*

Computational Physics,
Department of Physics and Astronomy,
Uppsala university, Sweden

August 31, 2022

*Electronic address: joelfbjervig@gmail.com

Introduction

In this part of the project, covering task 3 and 4, we use stochastic single-flip algorithms such as the Metropolis and Heat bath algorithm, which is applied to the Ising model. The ising model is used for a great many things, such as what we discuss here, namely simulating thermodynamical properties and phase transitions of a material of a square lattice structure.

Theory

The Monte Carlo Method

Integrals of four or lower dimensions, can preferably be determined with quadrature rule methods, such as the Simpsons or Bodes rule. Systems with a higher number of degrees of freedom becomes very costly for said methods since they done scale wall with increasing dimensions. This is a problem because many physical systems has a large number of DOF, such as quantum probability fields, N-particle models in condensed matter physics, or the many possible outcomes in a card game. In large The Monte Carlo method will approximate some integral by interpreting it as an expected outcome and randomly sample numbers to approximate this outcome, or integral of many variables.

The Metropolis Algorithm

The Metropolis algorithm is a first order Markov chain Monte Carlo method. It is of the first order because of its sampling dependence: the next step in the parameter space is only dependent on the current position, and not on it previous positions. Its classified under the Monte Carlo methods because its use of random numbers of some distribution. To further explain the concept of this algorithm, i will explain the computational steps:

1) Sample a value of θ_0 from some random distribution function $\theta_0 \sim \pi(\theta)$

2) For each iteration t propose a step to a new position in the parameter

$$\theta_t' \sim N(\theta_{t-1}, \sigma)$$

Where theta is sampled from some symmetric distribution, like normal or uniform, centered around θ_{t-1} with a standard deviation σ which is set manually. Symmetry of the distribution is needed because the probability density of a jump from θ_a θ_b and a jump from θ_b to θ_a has to be equal.

$$P(\theta_a, \theta_b) = P(\theta_b, \theta_a)$$

3) calculate the ratio r.

$$r = \frac{P(x|\theta'_t)P(\theta'_t)}{P(x|\theta'_{t-1})P(\theta'_{t-1})}$$

$P(a|b)$ means the likelihood of a happening, given b. r is then used to determine whether to move to a new position in the parameter space by comparing this ratio with a random sample of uniform distribution between zero and one.

$$\theta_t = \begin{cases} \theta'_t & \text{If} \quad r > u \sim U(0,1) \\ \theta_{t-1} & \text{Else} \end{cases}$$

The comparison criterion that dictates the next step will always be fulfilled if r is bigger than one, meaning that the new sampled probability is higher than the current one (when the numerator has a higher value than the denominator in the definition of r in step three).

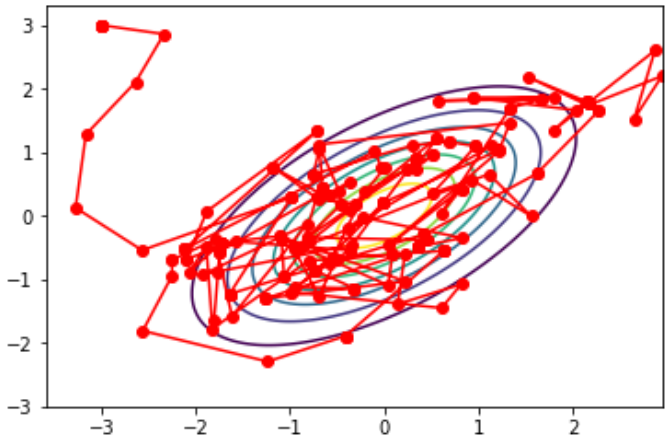
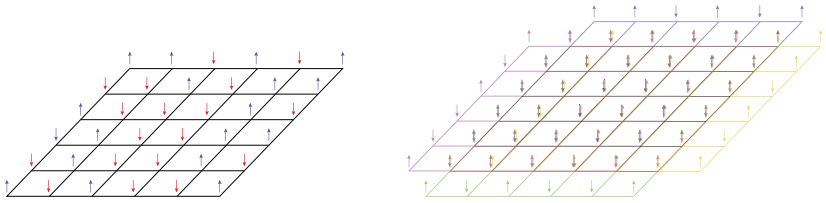


Figure 1: How the walk of the Metropolis algorithm could look like, for some two dimensional probability function. This is only for visual purposes, since a quadrature will perform well in approximating this integral

As seen in figure 1 the trajectory of the random walk will yield some large scale predictability. It is very likely that the algorithm will walk where the density function takes on a higher value, and will sometimes be allowed to take a step where the density function is less, depending on the comparison criteria with the uniformly sampled value u. As the number of steps grows towards infinity , the sampled distribution function of the random walk will converge toward the actual distribution function.

The Ising Model

The Ising model is not a physical system in of itself, but simply a square lattice where an element i can be one of two states $S_i \in \{-1, 1\}$, see figure 2. Because of its simplicity the model stands as foundation for a wide range of physical representations, mainly describing types of phase transitions, which occurs in many areas of physics. It is especially useful by the fact that its actually one of the few models that can be solved exactly.[Mat]



(a) An example configuration of the Ising model

(b) The sum of the four nearest neighbors by shifting of the lattice one element in each cardinal direction.

Figure 2: Realistically simulations have much larger lattices, but for pedagogical reasons the size here is 6×6.

Among superconductors, liquids and gases etc. that can undergo phase transitions, we chose to focus on the phenomenon of magnetization. Consider each element in the lattice to be an electron with some electron spin, $+1$ is spin up and -1 is spin down. It is not quite accurate since electron spins are of the size $1/2$ The spins has its own magnetic moment and on average all spins will contribute to a macro scale magnetization of the material it is part of. The more spins are aligned, the greater the magnetization is. [Mat] The Hamiltonian for the system is

$$H = -J \sum_{\langle \alpha\beta \rangle} S_\alpha S_\beta - B \sum_\alpha S_\alpha \tag{1}$$

The brackets $\langle \alpha\beta \rangle$ denotes a sum over the nearest neighbors pairs of spin (only in the four cardinal directions see figure **??**), and the strength of their interaction is determined by the scalar J(in the simulations made no external magnetic field will be present, meaning $B = 0$) The interaction poses a problem at the borders of the lattice, where no neighbors are present. Thus in a $N \times N$ lattice, the model requires a periodical boundary condition which creates a torodial shape see figure 3

$$\begin{cases} S_{(0,j)} = S_{N,j} \\ S_{(i,0)} = S_{(i,N)} \end{cases}$$

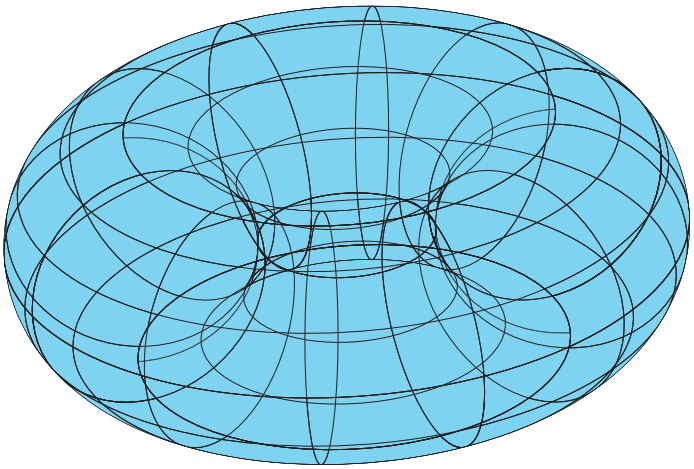


Figure 3: A toroid topology comes into fruition when periodic boundary conditions are imposed

Quantities to simulate

To get a better understanding of this system, there are several quantities to simulate. It is useful to consider the coupling strength J in units of temperature such that higher temperatures corresponds to a weaker coupling strength. To set up the system, the spin variable and the weighting is set up in a canonical form

$$\omega(S) = \frac{e^{-H(S)}}{Z} \qquad\qquad Z(J,B) = \sum_S e^{H(S)}$$

In task 3 three quantities are sought for, namely:

Magnetization - The total contribution of each electron spin in the lattice.

$$M = \frac{\partial \log Z}{\partial B} = \sum_S \omega(S) \left(\sum_\alpha S_\alpha \right) \tag{2}$$

Susceptibility is related to the fluctuations of the magnetization in the system as

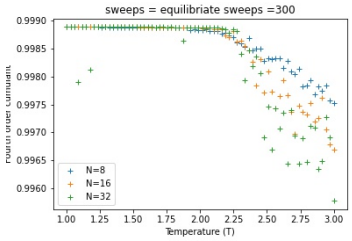
$$\chi = \frac{1}{k_B T} \left(\langle M^2 \rangle - \langle M \rangle^2 \right) \tag{3}$$

Specific heat at a constant field is similarly related to the energy fluctuations in the system as

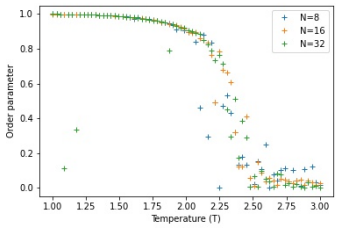
$$C_B = \frac{1}{k_B T^2} \left(\langle E^2 \rangle - \langle E \rangle^2 \right) \tag{4}$$

Where the energy E is obtained from the Hamiltonian of the system in equation 1 by summing over the nearest neighbors as described above. For clarification see the energy function in the code in appendix.

The simulation of these quantities as a function of the temperature T in the lattice are shown in the first four graphs in figures 4 and 5 below. The temperature is normalized and ranges from one to three. The system is ferromagnetic, meaning $0 < J$ and there is no external magnetic field applied

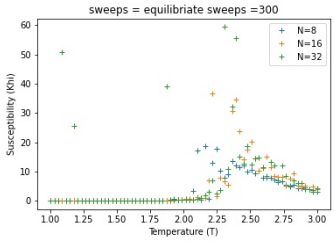


(a) The phase change temperature is at $T = 2.25$

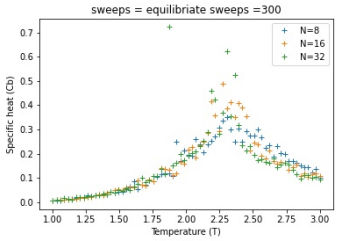


(b) The order parameter is a measure of the net-magnetization

Figure 4: At the phase change temperature we see a drastic change in the magnetization of the lattice as it



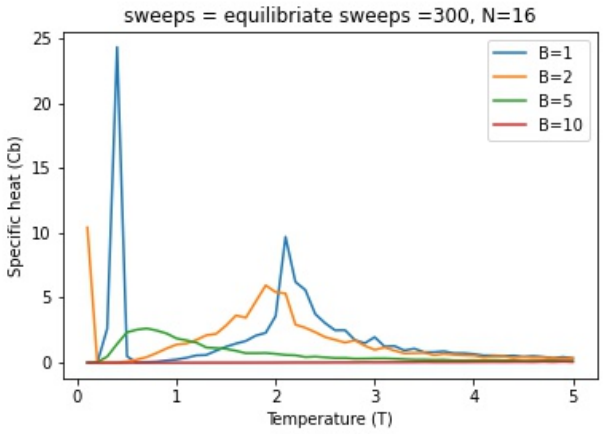
(a) Susceptibility



(b) Specific heat

Figure 5: Similarly, major changes occur at phase change temperature for the susceptibility and the specific heat of the material.

In part 4, instead a antiferromagnetic coupling is simulated, with a non-zero external magnetic field $B \neq 0$.



Discussion

The fourth order cumulant for a spin-1 Ising model is used to obtain the phase transition temperature. The cumulant can be seen in the graph to the left in figure 4 and it shows that the phase transition temperature is approximately $T = 2.25$. This value agrees with the other graphs as they all undergo significant change at that temperature. For example it is known that the specific heat of a material jumps to a relatively high value at a phase transition, which can be seen in the graph to the right in figure 4. If we look as the other figures **??** **??** we have the largest change of ordering of spins within the material, meaning the speed at which spins become disordered happens at the phase transition. If we take a look at the susceptibility, it seems to be the largest at the phase transition. Susceptibility is the measure of fluctuation of magnetization, and indeed the fluctuations of the magnetization are greater at that area in figure **??**. In the graphs for task 3 there are three plots, each representing a different lattice size: $N = \{8, 16, 32\}$. They all have the same behavior, but exhibits a noticeable and serious difference. Due to the random nature of Monte Carlo simulations, improbable configurations that deviates greatly from the expected behavior will occur at some point.These event will have a large effect on models with a small lattice size. Looking at the magnetization in figure 5, there are two configurations in the 32×32 lattice curve that has produced deviating values at $1 < T < 1.25$. However because of its lattice size, they are more improbable than for small lattice sizes. This conclusion is rather obvious since the computational scientists intuition is to simulate and redo experiments over a large population or a large amount of times.

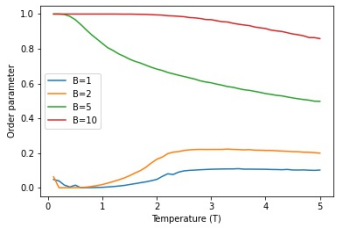
Now, consider the system set up for task 4: A 16×16 large lattice with antiferromagnetic coupling and an external magnetic field present. The first conclusion to be made from the left graph in figure **??** is that with a stronger external

field B, more spins will align to the field, meaning the order parameter is high. The plots are smooth and well behaved. While the susceptibility, showed on the right side in the figure, correlates well with the magnetization, it displays less precise results. But still one can see that when spins are subject to a force such that they align, they are also less likely to change spin. This can be seen in the graph as the magnetic field strength increases, the susceptibility decreases. Interestingly, the computed specific heat in figure behaves a bit strange, at least for $B = 1$ and $B = 2$ where they jump abruptly in the beginning at around $0 < T < 0.6$. This might simply be an artifact from the algorithms.

The last thing to discuss, is if one would use the Heat bath algorithm instead of the Metropolis algorithm. While they belong to the same category of *Singel Flip Algorithms*, The core difference between the two models lies in the way they determine to take the next step into a new state of the system θ_t'. As presented in the theory section above, the Metropolis algorithm always takes a step if the energy of that state is greater, otherwise it takes a step depending on the comparison of a sample from a uniform distribution. The heat bath however, does not split up the decision process into two cases, but will instead base its steps on one single acceptance ratio function which looks like this

$$y = \frac{e^{-\beta \Delta E}}{e^{-\beta \Delta E} + 1} \tag{5}$$

It is more apparent how the acceptance ratio looks like if the two acceptance functions are plotted on a graph, as in figure 7.



(a) mag p4

Comparing the two algorithms by simulating the order magnetization, the heat bath algorithm performs well in comparison, but shows some data points deviating at $2 < T < 2.5T$ in figure **??**.

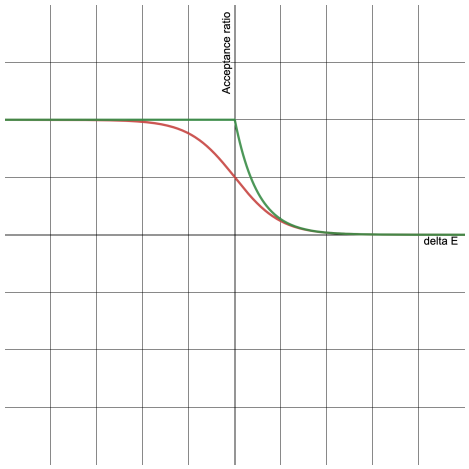


Figure 7: How the acceptance ratio differs for the Heat bath algorithm (red) and the Metropolis algorithm (green)

References

[Mat] Jacob Mattingleyl. *The Ising Model*. URL: `https://stanford.edu/~jeffjar/statmech/intro4.html`. (accessed: 24.03.2021).

1 Appendix

1.1 Task 3

```
import numpy as np
from numpy.random import rand
import matplotlib.pyplot as plt

###############
#Functions
###############

def initial(N):
    lattice = np.random.choice([-1,1],(N,N))
    return lattice


def metropolis(latt,temp):
    N=latt.shape[0]
    for i in range(N):
        for j in range(N):

                neigh = latt[(i+1)%N,j] + latt[i,(j+1)%N] + latt[(i-1)%N,j] + la
                dE = 2*latt[i,j]*neigh
                if dE < 0:
                    latt[i, j] *= -1
                elif rand() < np.exp(-dE/(kb*temp)):
                    latt[i, j] *= -1
    return latt


def energy(latt):
    #vectorial computation of the S_alpha*S_beta for the closest neighbours
    Sup = np.roll(latt, -1, axis=0)
    Sdown = np.roll(latt, 1, axis=0)
    Sleft = np.roll(latt, 1, axis=1)
    Sright = np.roll(latt, -1, axis=1)

    sum_neighbours= Sup+Sdown+Sleft+Sright

    H=-np.sum(sum_neighbours*latt)
    return H/4


def magnetization(config):
    mag = np.sum(config)
```

```
    return mag


################
#Main Code
################

def ising(N,T):

    E,M,Cb,Khi,U = np.zeros(T.shape), np.zeros(T.shape), np.zeros(T.shape), np.z


    norm1=sweeps*N**2
    norm2=sweeps**2*N**2

    for n in range(len(T)):
        e1 = m1 = e2 = m2 = m4 =0

        config = initial(N)

        for i in range(equilibrium): #loops to equilibrate the system
            metropolis(config, T[n]) #metropolis algorithm

        for i in range(sweeps):
            metropolis(config, T[n])

            En = energy(config)
            Mag = magnetization(config)

            e1+=En
            m1+=Mag
            m2+=Mag**2
            e2+=En**2
            m4+=Mag**4

        E[n] = e1/norm1
        M[n] = m1/norm1
        Cb[n] = (e2/norm1 - e1**2/norm2)/(kb*T[n]**2)
        Khi[n] = (m2/norm1 - m1**2/norm2)/(kb*T[n])
        U[n]=1-m4/(3*m2**2)

    return E,M,Cb,Khi,U

kb=1
J=1
```

```
equilibrium = 300
sweeps = 300
T= np.linspace(1, 3, 70)
N=np.array([8,16,32])

E = np.zeros((len(N),len(T)))
M,Cb,Khi,U = np.zeros(E.shape), np.zeros(E.shape), np.zeros(E.shape), np.zeros(E

for k in range(len(N)):
    E[k,:],M[k,:],Cb[k,:],Khi[k,:],U[k,:]=ising(N[k],T)

##############
#Plots
##############


plt.figure(1)
for i in range(len(N)):
    plt.plot(T,abs(M[i,:]),"+",label="N="+str(N[i]))
plt.legend()
plt.xlabel("Temperature (T)")
plt.ylabel("Order parameter")
plt.savefig("Magnetization.jpg")
plt.title("sweeps = equilibriate sweeps ="+str(sweeps))

plt.figure(2)
for i in range(len(N)):
    plt.plot(T,Khi[i,:],"+",label="N="+str(N[i]))
plt.legend()
plt.xlabel("Temperature (T)")
plt.ylabel("Susceptibility (Khi)")
plt.title("sweeps = equilibriate sweeps ="+str(sweeps))
plt.savefig("Susceptibility.jpg")

plt.figure(3)
for i in range(len(N)):
    plt.plot(T,Cb[i,:],"+",label="N="+str(N[i]))
plt.legend()
plt.xlabel("Temperature (T)")
plt.ylabel("Specific heat (Cb)")
plt.title("sweeps = equilibriate sweeps ="+str(sweeps))
plt.savefig("Specheat.jpg")

plt.figure(4)
for i in range(len(N)):
    plt.plot(T,U[i,:],"+",label="N="+str(N[i]))
```

```
plt.legend()
plt.xlabel("Temperature (T)")
plt.ylabel("Fourth order cumulant")
plt.title("sweeps = equilibriate sweeps ="+str(sweeps))
plt.savefig("Cumulant.jpg")
```

1.2 Task 4a

```
import numpy as np
from numpy.random import rand
import matplotlib.pyplot as plt

def initial(N):
    lattice = np.random.choice([-1,1],(N,N))
    return lattice


def metropolis(latt,temp,B):
    N=latt.shape[0]
    for i in range(N):
        for j in range(N):

                neigh = latt[(i+1)%N,j] + latt[i,(j+1)%N] + latt[(i-1)%N,j] + la
                dE = 2*J*latt[i,j]*neigh+2*latt[i,j]*B
                if dE < 0:
                    latt[i, j] *= -1
                elif rand() < np.exp(-dE/(kb*temp)):
                    latt[i, j] *= -1
    return latt


def energy(latt,B):
    #vectorial computation of the S_alpha*S_beta for the closest neighbours
    Sup = np.roll(latt, -1, axis=0)
    Sdown = np.roll(latt, 1, axis=0)
    Sleft = np.roll(latt, 1, axis=1)
    Sright = np.roll(latt, -1, axis=1)

    sum_neighbours= Sup+Sdown+Sleft+Sright

    H=-J*np.sum(sum_neighbours*latt)-B*np.sum(latt)
    return H


def magnetization(config):
    mag = np.sum(config)
    return mag

  def ising(N,T,B):

    E,M,Cb,Khi,U = np.zeros(T.shape), np.zeros(T.shape), np.zeros(T.shape), np.z
```

```
    norm1=sweeps*N**2
    norm2=sweeps**2*N**2

    for n in range(len(T)):
        e1 = m1 = e2 = m2 = m4 =0

        config = initial(N)

        for i in range(equilibrium): #loops to equilibrate the system
            config=metropolis(config, T[n],B) #metropolis algorithm

        for i in range(sweeps):
            config=metropolis(config, T[n],B)

            En = energy(config,B)
            Mag = magnetization(config)

            e1+=En
            m1+=Mag
            m2+=Mag**2
            e2+=En**2
            m4+=Mag**4
        E[n] = e1/norm1
        M[n] = m1/norm1
        Cb[n] = (e2/norm1 - e1**2/norm2)/(kb*T[n]**2)
        Khi[n] = (m2/norm1 - m1**2/norm2)/(kb*T[n])
        U[n]=1-m4/(3*m2**2)

    return E,M,Cb,Khi,U

  kb=1
J=-1

equilibrium = 300
sweeps = 300

T= np.linspace(0.1, 5, 50)
N=16
B=np.array([1,2,5,10])

E = np.zeros((len(B),len(T)))
M,Cb,Khi,U = np.zeros(E.shape), np.zeros(E.shape), np.zeros(E.shape), np.zeros(E

for k in range(len(B)):
    E[k,:],M[k,:],Cb[k,:],Khi[k,:],U[k,:]=ising(N,T,B[k])
```

```
plt.figure(1)
for i in range(len(B)):
    plt.plot(T,abs(M[i,:]),label="B="+str(B[i]))
plt.legend()
plt.xlabel("Temperature (T)")
plt.ylabel("Order parameter")
plt.savefig("P4_Magnetization.jpg")
plt.title("sweeps = equilibriate sweeps ="+str(sweeps)+", N="+str(N))

plt.figure(2)
for i in range(len(B)):
    plt.plot(T,Khi[i,:],label="B="+str(B[i]))
plt.legend()
plt.xlabel("Temperature (T)")
plt.ylabel("Susceptibility (Khi)")
plt.title("sweeps = equilibriate sweeps ="+str(sweeps)+", N="+str(N))
plt.savefig("P4_suscept.jpg")

plt.figure(3)
for i in range(len(B)):
    plt.plot(T,Cb[i,:],label="B="+str(B[i]))
plt.legend()
plt.xlabel("Temperature (T)")
plt.ylabel("Specific heat (Cb)")
plt.title("sweeps = equilibriate sweeps ="+str(sweeps)+", N="+str(N))
plt.savefig("P4_specheat.jpg")
```

1.3 Task 4b

```
import numpy as np
from numpy.random import rand
import matplotlib.pyplot as plt

def initial(N):
    lattice = np.random.choice([-1,1],(N,N))
    return lattice


def heatbath(latt,temp):
    N=latt.shape[0]
    for i in range(N):
        for j in range(N):

                neigh = latt[(i+1)%N,j] + latt[i,(j+1)%N] + latt[(i-1)%N,j] + la
                arg=2*J*neigh/(kb*temp)
                p=np.exp(arg)/(1+np.exp(arg))

                if rand()<p:
                    latt[i,j]=1
                else:
                    latt[i,j]=-1
    return latt

def metropolis(latt,temp):
    N=latt.shape[0]
    for i in range(N):
        for j in range(N):

                neigh = latt[(i+1)%N,j] + latt[i,(j+1)%N] + latt[(i-1)%N,j] + la
                dE = 2*J*latt[i,j]*neigh+2*latt[i,j]*B
                if dE < 0:
                    latt[i, j] *= -1
                elif rand() < np.exp(-dE/(kb*temp)):
                    latt[i, j] *= -1
    return latt

def energy(latt,B):
    #vectorial computation of the S_alpha*S_beta for the closest neighbours
    Sup = np.roll(latt, -1, axis=0)
    Sdown = np.roll(latt, 1, axis=0)
    Sleft = np.roll(latt, 1, axis=1)
    Sright = np.roll(latt, -1, axis=1)
```

```
    sum_neighbours= Sup+Sdown+Sleft+Sright

    H=-J*np.sum(sum_neighbours*latt)-B*np.sum(latt)
    return H


def magnetization(config):
    mag = np.sum(config)
    return mag


  def ising(N,T,B,algo):

    E,M,Cb,Khi,U = np.zeros(T.shape), np.zeros(T.shape), np.zeros(T.shape), np.z


    norm1=sweeps*N**2
    norm2=sweeps**2*N**2

    for n in range(len(T)):
        e1 = m1 = e2 = m2 = m4 =0

        config = initial(N)

        for i in range(equilibrium): #loops to equilibrate the system
            config=algo(config, T[n])

        for i in range(sweeps):
            config=algo(config, T[n])

            En = energy(config,B)
            Mag = magnetization(config)

            e1+=En
            m1+=Mag
            m2+=Mag**2
            e2+=En**2
            m4+=Mag**4
        E[n] = e1/norm1
        M[n] = m1/norm1
        Cb[n] = (e2/norm1 - e1**2/norm2)/(kb*T[n]**2)
        Khi[n] = (m2/norm1 - m1**2/norm2)/(kb*T[n])
        U[n]=1-m4/(3*m2**2)

    return E,M,Cb,Khi,U
```

```
   kb=1
J=1
B=0
equilibrium = 300
sweeps = 300

T= np.linspace(1, 3.5, 50)
N=16


Em = np.zeros(T.shape)
Mm,Cbm,Khim,Um = np.zeros(T.shape), np.zeros(T.shape), np.zeros(T.shape), np.zer


Eh = np.zeros(T.shape)
Mh,Cbh,Khih,Uh = np.zeros(T.shape), np.zeros(T.shape), np.zeros(T.shape), np.zer


Em,Mm,Cbm,Khim,Um=ising(N,T,B,metropolis)
Eh,Mh,Cbh,Khih,Uh=ising(N,T,B,heatbath)

plt.figure(1)
plt.plot(T,abs(Mm),'ro',label="metropolis")
plt.plot(T,abs(Mh),'co',label="heatbath")
plt.legend()
plt.xlabel("Temperature (T)")
plt.ylabel("Order parameter")
plt.title("sweeps = equilibriate sweeps ="+str(sweeps)+", N="+str(N))
plt.savefig("MagnetP4b.jpg")

plt.figure(2)
plt.plot(T,Khim,'ro',label="metropolis")
plt.plot(T,Khih,'co',label="heatbath")
plt.legend()
plt.xlabel("Temperature (T)")
plt.ylabel("Susceptibility (Khi)")
plt.title("sweeps = equilibriate sweeps ="+str(sweeps)+", N="+str(N))
plt.savefig("SuceptP4b.jpg")

plt.figure(3)
plt.plot(T,Cbm,'ro',label="metropolis")
plt.plot(T,Cbh,'co',label="heatbath")
plt.legend()
plt.xlabel("Temperature (T)")
plt.ylabel("Specific heat (Cb)")
```

```
plt.title("sweeps = equilibriate sweeps ="+str(sweeps)+" , N="+str(N))
plt.savefig("SpeheatP4b.jpg")
```