Ising Model in Python

Jospeh Latta

April 27, 2018

1 Introduction

Ferromagnetic materials exhibit the ability to obtain and maintain magnetization even in the absence of an external magnetic field. This ferromagnetism arises due to electron magnetic moments and exchange interactions. Each of these effects are quantum mechanical and thus the ferromagnetic system is an excellent candidate for Monte Carlo simulation due to the statistical nature of this process. [1] The electron magnetic moment is the magnetic moment of the electron due to its angular momentum and electric charge. The exchange interaction is the interaction between electron moments which arises due to the presence of unpaired electrons and their spins. This interaction creates an energy difference between the spin up and spin down states of the electron dipoles. Locally, electron spins tend to align in the same direction which reduces their electric repulsion . Ferromagnetics exhibit the ability for the dipoles across the material to align simultaneously causing spontaneous magnetization even in the absence of an external magnetic field. The thermal motion of dipoles can compete with this alignment process. Because of this, as temperature increases to a critical temperature, called the Curie temperature, the material loses its ability to maintain magnetization.

2 Model

To model the ferromagnetic material, a square lattice is created filled with a random distribution of -1 or 1 spin states. The energy of a given spin configuration is given by

$$H(\sigma) = \sum_{i,j} J_{ij}\sigma_i\sigma_j - \mu \sum_i h_i\sigma_i \tag{1}$$

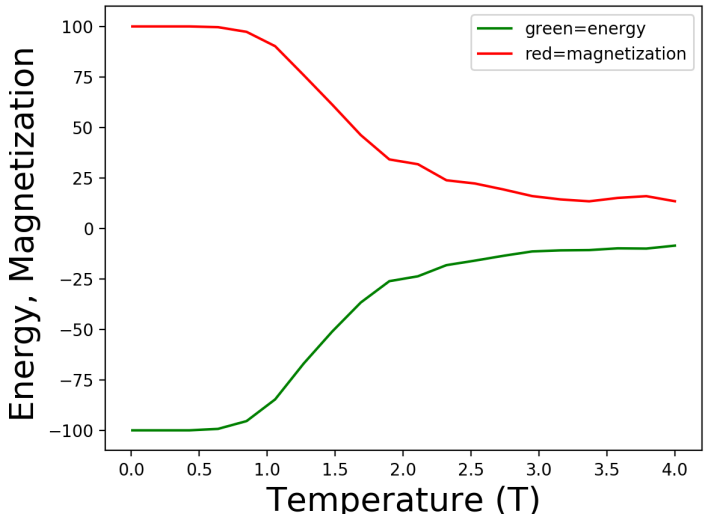
Here, σ represents the spin state of the i or j lattice position, J is the interaction term between adjacent dipoles, μ is the magnetic moment, and h is the magnetic field at site i. In the absence of a magnetic field, the second term of the energy equation goes away.

To model this, the Metropolis-Hastings algorithm was used. This algorithm uses random numbers to choose a lattice site, then calculates the energy change if the spin at that lattice site were flipped. [2]If the resulting energy from the spin flip is lower than the previous energy, the flip is accepted. If the energy is higher, the flip is accepted based on the following probability.

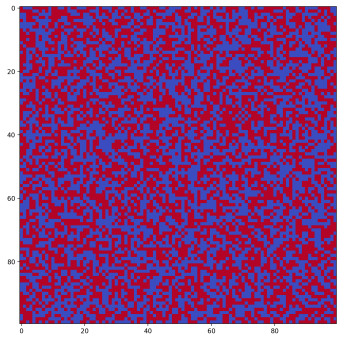
$$P = e^{\frac{H(\sigma)-H(\sigma')}{kT}} \tag{2}$$

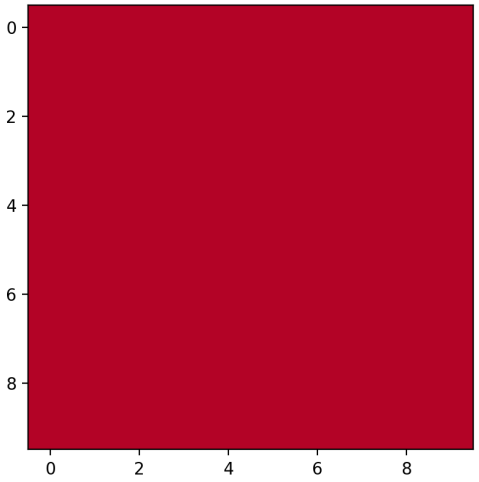
Here, k=Boltzmann constant and T=Temperature. This results in the probability of reaching higher energy configurations increases as temperature increases, but the system as a whole will always be moving into a lower energy state.

In the python program shown in Listing 1, this model is implemented to plot the system energy and magnetization as a function of increasing temperature. Figure 1 shows the resulting plot which indicates that the Curie temperature is reached at 1.35 degrees C (the inflection point).



Next the program simulates a ferromagnetic material at constant temperature with no external magnetic field in order to show simultaneous magnetization. Figure 2 shows the system at the beginning and figure 3 shows the system after full magnetization has occurred.





There are many factors that affect the accuracy of this model. The largest factor is the lattice size. For the NxN lattice, the number of possible states is 2^N so the larger lattice the more accurate the model. In addition to this, a larger number of Monte Carlo steps as well as temperature steps will increase the accuracy of the calculations but also has significant impact on the run time of the code.

Listing 1: Python code

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

```
N = 10 # square lattice size
shape = (N,N) # set up NxN lattice
spin = np.random.choice([-1,1], size=shape) # random spin cofiguration of -1 or
ExField = np.full(shape,0) # fill External magnetic field with 0's
tempStep = 20 # temperature steps
MCstep = 100 # Monte Carlo steps
T = np.linspace(0.01, 4, tempStep) # temperature range for Energy and Magnetizat
simTemp = 0.1 # constant temp for spin simulation

MCEnergy = np.zeros(tempStep) # initialize Energy array
MCMagnetization = np.zeros(tempStep) # initialize Magnetization array

### MCmetro utilizes the Metropolis algorithm to determine spin:
### -if flipping spin results in Energy decrease, it is flipped
### -if flipping spin results in constant or increased Energy, the probability i
###  flipped = exp(-deltaE/kT)

def MCmetro(spin, beta):

    for i in range(N):

        for j in range(N):

                rand1 = np.random.randint(0, N)
                rand2 = np.random.randint(0, N)
                detSpin =  spin[rand1, rand2]
                met = spin[(rand1+1)%N,rand2] + spin[rand1,(rand2+1)%N] + spin[(
                metroE = 2*detSpin*met

                if metroE < 0:
                    detSpin *= -1

                elif rand() < np.exp(-met*beta):
                    detSpin *= -1

                spin[rand1, rand2] = detSpin

    return spin

### Energy determines the energy at a determined spin configuration

def Energy(spin):

    energy = 0
```

```
    l = len(spin)

    for i in range(l):

        for j in range(l):

            updateSpin = spin[i,j]
            met = spin[(i+1)%N, j] + spin[i,(j+1)%N] + spin[(i-1)%N, j] + spin[i
            energy += -met*updateSpin

    return energy/4.0

### Magnetization determines the magnetization at a determined spin configuratio
def Magnetization(spin):

    mag = np.sum(spin)

    return mag

### simMetro utilizes the Metropolis algorithm to determine energy configuration

def simMetro(deltaE, simTemp):

  return np.exp((deltaE) / simTemp)

### animate updates spin configuration based on Monte Carlo to simulate magnetiz

def animate(spin, simTemp):

  spin1 = np.copy(spin)
  randA = np.random.randint(spin.shape[0])
  randB = np.random.randint(spin.shape[1])
  spin1[randA, randB] *= -1
  energyNow = Energy(spin)
  energyUpdate = Energy(spin1)
  deltaE = energyNow - energyUpdate

  if  simMetro(deltaE, simTemp) > np.random.random():

    return spin1

  else:

    return spin

### mainPlot uses Energy and Magnetization functions to plot as a function of te
```

```
def mainPlot():
    for m in range(len(T)):
        E1 = M1 = 0

        for i in range(MCstep):
            MCmetro(spin, 1.0/T[m])

        for i in range(MCstep):
            MCmetro(spin, 1.0/T[m])
            Ene = Energy(spin)
            Mag = Magnetization(spin)

            E1 = E1 + Ene
            M1 = M1 + Mag

            MCEnergy[m]          = E1/(N**2)
            MCMagnetization[m]   = M1/(N**2)

    plt.plot(T, MCEnergy, 'g',label = 'green=energy')
    plt.plot(T, abs(MCMagnetization), 'r',label = 'red=magnetization')
    plt.legend()
    plt.xlabel("Temperature (T)", fontsize=20)
    plt.ylabel("Energy , Magnetization ", fontsize=20)
    plt.show()

mainPlot()

## Animation code ###
plt.show()

im = plt.imshow(spin, cmap='coolwarm', vmin=-1, vmax=1, interpolation='none')
t = 0
while True:
  if t % 10 == 0:
    im.set_data(spin)
    plt.draw()
  spin = animate(spin, simTemp)
  plt.pause(1e-10)
  t += 1
```

References

[1] Alexey Khorev. A monte carlo implementation of the ising model in python. *Unknown*, Aug 2017.

[2] M. E. J. Newman. *Computational physics.* Createspace, 2013.