```
1
2   # coding: utf-8
3
4   # In[ ]:
5
6
7   # Import modules
8   import numpy as np
9   import matplotlib.pyplot as plt
10  from scipy import optimize
11  import time
12
13
14  # In[ ]:
15
16
17  # Class representing a 2D Ising model
18  class Ising:
19      def __init__(self, N, J, H):
20          self.lattice = np.random.choice([-1,1], size=(N,N)) # initial state
21          self.N = N # dimension
22          self.J = np.abs(J) # interaction energy, non-negative by definition
23          self.H = H # external field
24
25
26  # In[ ]:
27
28
29  # Function to calculate energy change when spin(i,j) is flipped, assuming periodic
    boundary conditions
30  def energy_change_of_flip(state, pos_i, pos_j):
31      return
        2*state.lattice[pos_i,pos_j]*(state.J*(state.lattice[pos_i,(pos_j+1)%state.N] +
        state.lattice[(pos_i-1)%state.N,pos_j] + state.lattice[(pos_i+1)%state.N,pos_j] +
        state.lattice[pos_i,(pos_j-1)%state.N])+state.H)
32
33
34  # In[ ]:
35
36
37  # Monte Carlo sweep using Metropolis algorithm
38  def monte_carlo_sweep(state, temperature):
39      for i in range(len(state.lattice)**2):
40          # pick a spin randomly and calculate energy change if it's flipped
41          pos_i = np.random.randint(0,len(state.lattice))
42          pos_j = np.random.randint(0,len(state.lattice))
43          delta_E = energy_change_of_flip(state,pos_i,pos_j)
44
45          # to flip or not to flip?
46          if delta_E < 0 or np.exp(-delta_E/temperature) > np.random.random():
47              state.lattice[pos_i,pos_j] *= -1
48
49
50  # In[ ]:
51
52
53  # Function to calculate magnetisation
54  def calculate_magnetisation(state):
55      return np.sum(state.lattice)
56
57
58  # In[ ]:
59
60
61  # Function to calculate energy
62  def calculate_energy(state):
63      interaction_energy = 0.0
```