# PH5730 - Assignment 1 (BONUS)

Himanshu Rajnish Borkar - EE22B070

March 2, 2025

Old submission had an error in the code and plot, this is the corrected code and plot.

## 1 Theory: Triangular Lattice

The energy (restriction) of a lattice system is represented by:

$$E = J \sum_{\langle i,j \rangle} s_i s_j \tag{1}$$

The slight difference between Square and Triangular lattice is that we can transform the triangular lattice into a square lattice with each element of the lattice having 2 more nearest neighbors in the following configuration transformation:

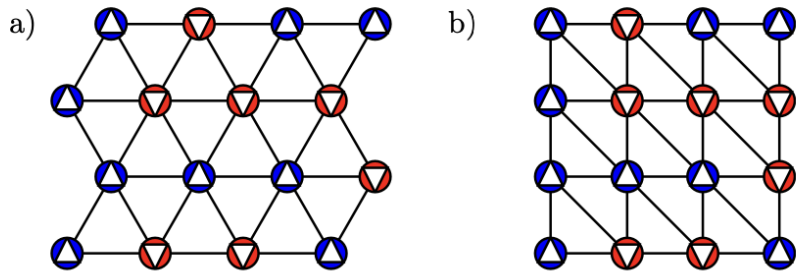


**Figure 1:** (a) Triangular Lattice (b) Linear transformation of the lattice to a square lattice (ref: Classical Monte Carlo, M-LAB)

## 2 Anti-Ferromagnetic Triangular Lattice Code (J = +1):

The following Python code simulates the Ising Model for Anti-ferromagnetic Triangular lattice:

```
# Antiferromagnetic

import numpy as np
import matplotlib.pyplot as plt

# Lattice generation
s_ = [1, -1]
N = 10  # Lattice size

lattice = np.random.choice(s_, size=(N, N))

lower_temp = 0.5
upper_temp = 7
temp_pts = 200
t_limit = 2000
limit = 9000
```

```
17
18  # Restriction Calc updated
19
20  def energyCalc(lattice, N):
21      ener = 0
22      for i in range(N):
23          for j in range(N):
24              s = lattice[i,j]
25              nb = lattice[(i+1)%N,j] + lattice[i,(j+1)%N] + lattice[(i-1)%N,j] + lattice[i,(j
                    -1)%N] + lattice[(i+1)%N,(j+1)%N] + lattice[(i-1)%N,(j-1)%N]
26              ener += s*nb
27      return ener
28
29  #Magnetization Calculations
30
31  def magCalc(lattice):
32      mag = np.sum(lattice)
33      return mag
34
35  #Monte Carlo Steps - Metropolis Algorithm
36
37  def mc_steps(lattice, beta):
38      for i in range(N):
39          (a,b) = np.random.randint(0, N, size=2)
40          s = lattice[a,b]
41          n_b = lattice[(a+1)%N,b] + lattice[a,(b+1)%N] + lattice[(a-1)%N,b] + lattice[a,(b-1)
                %N] + lattice[(a+1)%N,(b+1)%N] + lattice[(a-1)%N,(b-1)%N]
42          delta_restriction = -2*s*n_b
43          if delta_restriction < 0:
44            s *= -1
45          elif np.random.rand() < np.exp(-delta_restriction*beta):
46            s *= -1
47          lattice[a,b] = s
48      return lattice
49
50  Temperatures = np.linspace(lower_temp,upper_temp,temp_pts)
51
52  energy_arr = []
53  mag_arr = []
54  C_arr = []
55  X_arr = []
56
57  t_limit = 1000
58  limit = 9000
59  for temp in Temperatures:
60      ener = 0
61      magn = 0
62      ener2 = 0
63      magn2 = 0
64
65      l = np.copy(lattice)
66      beta = 1/temp
67      for t in range(t_limit):
68          mc_steps(l,beta)
69      for z in range(limit):
70          mc_steps(l,beta)
71          e = energyCalc(l,N)
72          m = magCalc(l)
73
74          ener += e
75          ener2 += e**2
76          magn += m
77          magn2 += m**2
78
79      energy_arr.append(ener/(N*N*limit))
80      C_arr.append(((ener2/(N*N*limit) - ((ener*ener)/(N*N*limit*limit)))*(beta**2)))
81      mag_arr.append(magn/(N*N*limit))
82      X_arr.append(((magn2/(N*N*limit) - ((magn*magn)/(N*N*limit*limit)))*(beta)))
```

```
83
84  f = plt.figure(figsize=(18, 10))
85
86  sp =  f.add_subplot(2, 2, 1 )
87  plt.scatter(Temperatures, energy_arr, s=10, marker='o', color='red')
88  plt.xlabel("Temperature (T)")
89  plt.ylabel("Energy ")
90  plt.axis('tight')
91
92  sp =  f.add_subplot(2, 2, 3 )
93  plt.scatter(Temperatures, C_arr, s=10, marker='o', color='blue')
94  plt.xlabel("Temperature (T)")
95  plt.ylabel("Specific heat ")
96  plt.axis('tight')
97
98  sp =  f.add_subplot(2, 2, 2 )
99  plt.scatter(Temperatures, np.abs(mag_arr), s=10, marker='o', color='green')
100 plt.xlabel("Temperature (T)")
101 plt.ylabel("Magnetization ")
102 plt.axis('tight')
103
104 sp =  f.add_subplot(2, 2, 4 )
105 plt.scatter(Temperatures, X_arr, s=10, marker='o', color='black')
106 plt.xlabel("Temperature (T)")
107 plt.ylabel("Magnetic Susceptibility ")
108 plt.axis('tight')
```