Submitted by: Anushka Menon and Yashasvee Goel

Answer 1

For any two sites i,j the interaction is termed as J.

The role of J is as follows:

J>0 → Interaction is ferromagnetic

J<0 → Interaction is anti-ferromagnetic

J=0 → Non-interacting

Answer 2

When we consider periodic boundary conditions, the last spins are considered to be their nearest neighbour of each other.

Answer 3

```
# Defining the necessary modules

import matplotlib.pyplot as plt
import numpy as np
from numpy.random import random
from random import choice

# The 1-D Ising Model

def H(J,h,s):   # Hamiltonian
    ret_value = 0.0
    n = len(s)
    for x in range(n):
#          We start by iterating over all spin states
        ret_value += -J*(s[(x-1)%n]*s[x])-h*s[x] # for periodic
boundary condition
    return ret_value

def S_random(n):
#      Defining array of length n filled with +1 or -1 random spin
states and appending it to S
    s = []
    for x in range(N):
        s.append(choice((+1,-1)))
    return s

# We have considered J = 1 and k_B = 1 as suggested in the question
J=1

# After differentiating the expression for <m> analytically, we get
```

```
the following expression:

def m_exact(n,h,T):
    # ret_m = (np.tanh(h/T))/n
    # return float(ret_m)
    sin = np.sinh(h)
    cos = np.cosh(h)
    exp = np.exp(-4*J)
    ratio = (cos-np.sqrt(sin**2+exp))/(cos+np.sqrt(sin**2+exp))
    ret_value = (1-ratio**N)/(1+ratio**N)
    ret_value *= sin/np.sqrt(sin**2+exp)
    return ret_value



def m_aprox(n,h,T,I):
    num = 0.0
    den = 0.0
    for i in range(I):
        S = S_random(N)                             # create random
configuration
        num += (sum(S)/N)*np.exp(-H(J,h,S)/T)  # use the observable
sum(spin) / #of spins
        den += np.exp(-H(J,h,S)/T)
    return num/den
```

The relevant dimensionless ratios are as follows:

(i) J/T and,

(ii) h/T

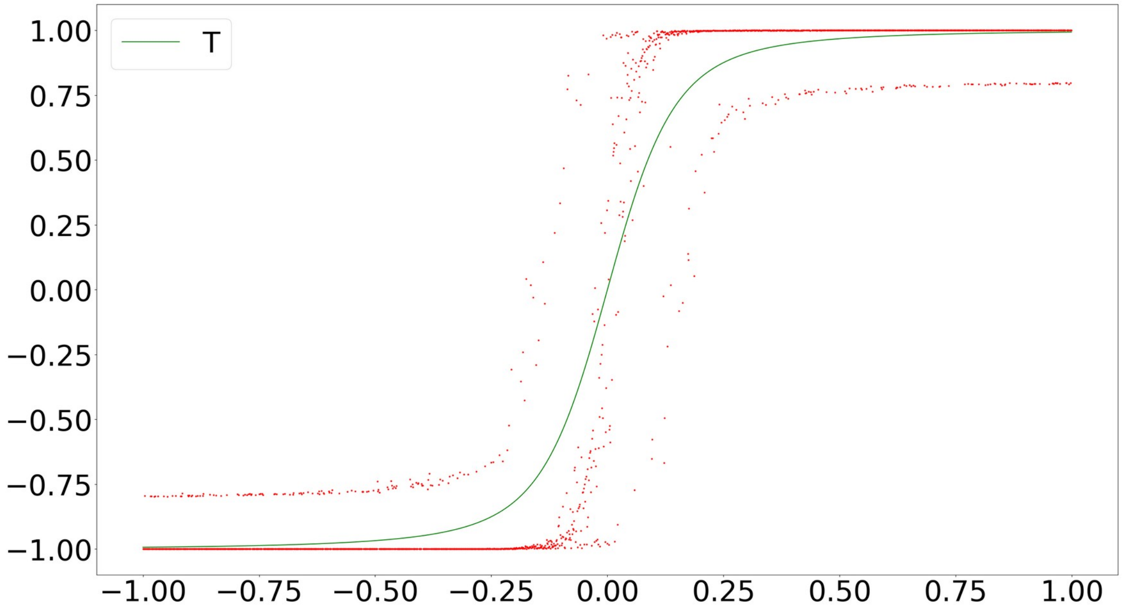
¿ m>¿ vs h for fixed N

```
plt.figure(figsize=(32,18))
plt.rc('font', size=50)

N = 10                          # Number of lattice sites           <= ~20
I  = int(2**(N+1))              # Number of generated spin configurations

# The range of h is given as [-1,1]

hRange = np.arange(-1 ,1,0.001)

exactVal=[m_exact(N,h,0.5)for h in hRange]
aproxVal=[m_aprox(N,h,0.5,I)for h in hRange]
```

```
aproxError=[val*I**-0.5 for val in aproxVal]

plt.plot(hRange ,exactVal ,'r',color="green")
# We have chosen the value of T as 0.5
plt.legend(("T = 0.5"))

plt.errorbar(hRange, aproxVal, yerr=aproxError, fmt='ro',
markersize=2)

plt.ylabel('<m>')
plt.xlabel('h')
plt.title("Graph for <m> vs h for fixed N")
plt.show()
```

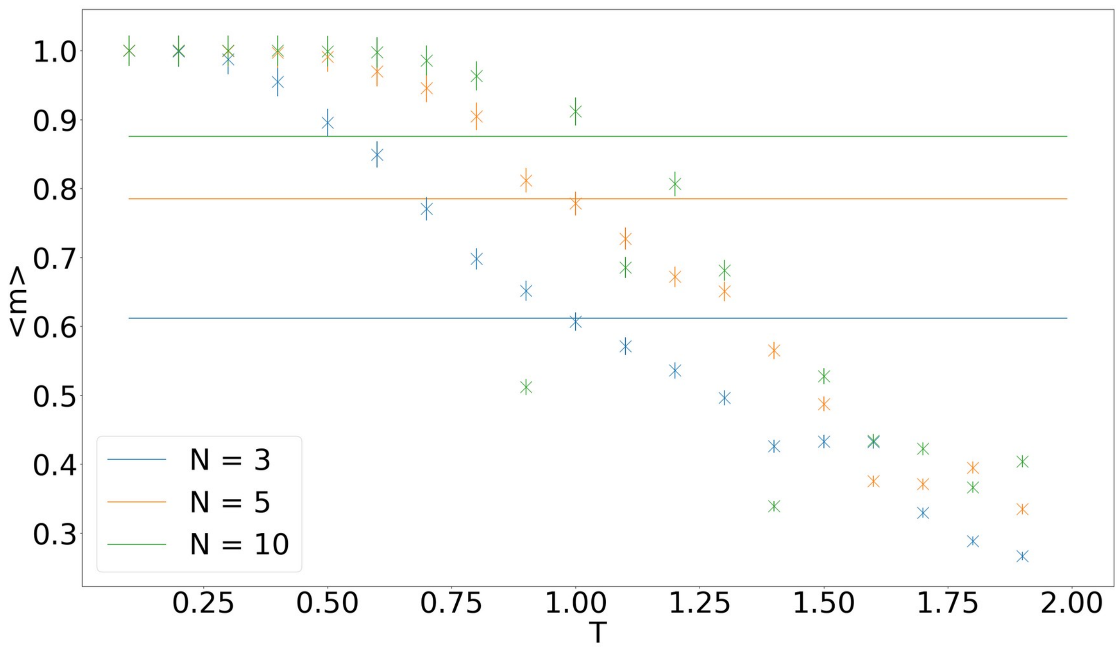
```
/tmp/ipykernel_7392/3058133422.py:16: UserWarning: color is
redundantly defined by the 'color' keyword argument and the fmt string
"r" (-> color=(1.0, 0.0, 0.0, 1)). The keyword argument will take
precedence.
  plt.plot(hRange ,exactVal ,'r',color="green")

----------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In [6], line 20
     17 # We have chosen the value of T as 0.5
     18 plt.legend(("T = 0.5"))
---> 20 plt.errorbar(hRange, aproxVal, yerr=aproxError, fmt='ro',
markersize=2)
     22 plt.ylabel('<m>')
     23 plt.xlabel('h')

File ~/.local/lib/python3.8/site-packages/matplotlib/pyplot.py:2482,
in errorbar(x, y, yerr, xerr, fmt, ecolor, elinewidth, capsize,
barsabove, lolims, uplims, xlolims, xuplims, errorevery, capthick,
data, **kwargs)
   2476 @_copy_docstring_and_deprecators(Axes.errorbar)
   2477 def errorbar(
   2478         x, y, yerr=None, xerr=None, fmt='', ecolor=None,
   2479         elinewidth=None, capsize=None, barsabove=False,
lolims=False,
   2480         uplims=False, xlolims=False, xuplims=False,
errorevery=1,
   2481         capthick=None, *, data=None, **kwargs):
-> 2482     return gca().errorbar(
   2483         x, y, yerr=yerr, xerr=xerr, fmt=fmt, ecolor=ecolor,
   2484         elinewidth=elinewidth, capsize=capsize,
barsabove=barsabove,
```

```
   2485         lolims=lolims, uplims=uplims, xlolims=xlolims,
   2486         xuplims=xuplims, errorevery=errorevery,
capthick=capthick,
   2487         **({"data": data} if data is not None else {}),
**kwargs)

File ~/.local/lib/python3.8/site-packages/matplotlib/__init__.py:1423,
in _preprocess_data.<locals>.inner(ax, data, *args, **kwargs)
   1420 @functools.wraps(func)
   1421 def inner(ax, *args, data=None, **kwargs):
   1422     if data is None:
-> 1423         return func(ax, *map(sanitize_sequence, args),
**kwargs)
   1425     bound = new_sig.bind(ax, *args, **kwargs)
   1426     auto_label = (bound.arguments.get(label_namer)
   1427                   or bound.kwargs.get(label_namer))

File
~/.local/lib/python3.8/site-packages/matplotlib/axes/_axes.py:3587, in
Axes.errorbar(self, x, y, yerr, xerr, fmt, ecolor, elinewidth,
capsize, barsabove, lolims, uplims, xlolims, xuplims, errorevery,
capthick, **kwargs)
   3584 res = np.zeros(err.shape, dtype=bool)  # Default in case of
nan
   3585 if np.any(np.less(err, -err, out=res, where=(err == err))):
   3586     # like err<0, but also works for timedelta and nan.
-> 3587     raise ValueError(
   3588         f"'{dep_axis}err' must not contain negative values")
   3589 # This is like
   3590 #     elow, ehigh = np.broadcast_to(...)
   3591 #     return dep - elow * ~lolims, dep + ehigh * ~uplims
   3592 # except that broadcast_to would strip units.
   3593 low, high = dep + np.row_stack([-(1 - lolims), 1 - uplims]) *
err

ValueError: 'yerr' must not contain negative values
```



¿m>¿ vs N for fixed h

```
plt.figure(figsize=(32,18))
plt.rc('font', size=50)

h = 0.25                  # Fixed external magnetic field
I  = int(2**(N+1))       # Number of generated spin configurations

NRange = [3,5,10]        # Number of lattice sites


TExactRange = np.arange(0.1 ,2,0.01)  #For a range of Temperature
TAproxRange = np.arange(0.1 ,2,0.1)

for N in NRange:
    exactVal=[m_exact(N,h,T)for T in TExactRange]
    plt.plot(TExactRange ,exactVal)
plt.legend(("N = 3", "N = 5", "N = 10"))

plt.gca().set_prop_cycle(None) #reset colour cycle

for N in NRange:
    aproxVal=[m_aprox(N,h,T,I)for T in TAproxRange]
    aproxError=[val*I**-0.5 for val in aproxVal]
    plt.errorbar(TAproxRange, aproxVal, yerr=aproxError, fmt = 'x', 
markersize=20)
plt.ylabel('<m>')
plt.xlabel('T')
plt.show()
```



1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
<m>
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
T
N = 3
N = 5
N = 10