```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Importing useful libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
np.random.seed(42)
```

Preparing the simulation

Ising model parameters

```
N = 20   # number of spins in one dimension
J = 1.0    # coupling strength
T = np.array([Temp for Temp in np.arange(0.05,10.05,0.05)]\
             +[Temp for Temp in np.arange(10.5,40.5,0.5)])# temperature in Kelvin
H = 0.0    # external field
k = 1 # Boltzmann constant in J/K
beta = (k*T)**-1  # inverse temperature
```

```
print(len(T),T)
```

```
260 [ 0.05  0.1   0.15  0.2   0.25  0.3   0.35  0.4   0.45  0.5   0.55  0.6
   0.65  0.7   0.75  0.8   0.85  0.9   0.95  1.    1.05  1.1   1.15  1.2
   1.25  1.3   1.35  1.4   1.45  1.5   1.55  1.6   1.65  1.7   1.75  1.8
   1.85  1.9   1.95  2.    2.05  2.1   2.15  2.2   2.25  2.3   2.35  2.4
   2.45  2.5   2.55  2.6   2.65  2.7   2.75  2.8   2.85  2.9   2.95  3.
   3.05  3.1   3.15  3.2   3.25  3.3   3.35  3.4   3.45  3.5   3.55  3.6
   3.65  3.7   3.75  3.8   3.85  3.9   3.95  4.    4.05  4.1   4.15  4.2
   4.25  4.3   4.35  4.4   4.45  4.5   4.55  4.6   4.65  4.7   4.75  4.8
   4.85  4.9   4.95  5.    5.05  5.1   5.15  5.2   5.25  5.3   5.35  5.4
   5.45  5.5   5.55  5.6   5.65  5.7   5.75  5.8   5.85  5.9   5.95  6.
   6.05  6.1   6.15  6.2   6.25  6.3   6.35  6.4   6.45  6.5   6.55  6.6
   6.65  6.7   6.75  6.8   6.85  6.9   6.95  7.    7.05  7.1   7.15  7.2
   7.25  7.3   7.35  7.4   7.45  7.5   7.55  7.6   7.65  7.7   7.75  7.8
   7.85  7.9   7.95  8.    8.05  8.1   8.15  8.2   8.25  8.3   8.35  8.4
   8.45  8.5   8.55  8.6   8.65  8.7   8.75  8.8   8.85  8.9   8.95  9.
   9.05  9.1   9.15  9.2   9.25  9.3   9.35  9.4   9.45  9.5   9.55  9.6
   9.65  9.7   9.75  9.8   9.85  9.9   9.95 10.   10.5  11.   11.5  12.
  12.5  13.   13.5  14.   14.5  15.   15.5  16.   16.5  17.   17.5  18.
  18.5  19.   19.5  20.   20.5  21.   21.5  22.   22.5  23.   23.5  24.
  24.5  25.   25.5  26.   26.5  27.   27.5  28.   28.5  29.   29.5  30.
  30.5  31.   31.5  32.   32.5  33.   33.5  34.   34.5  35.   35.5  36.
  36.5  37.   37.5  38.   38.5  39.   39.5  40.  ]
```

Defining Required methods

```
def NNInteraction_energy(Ising,periodicity=True):
  '''
  This function calculates the nearest neighbour spin interaction energy of the 3D Ising model
  '''
  energy = 0
  if periodicity: # periodic boundary condition
    for i in range(N):
      for j in range(N):
        for k in range(N):
          for a in [-1,1]:energy += -1*J* Ising[i][j][k]*( Ising[(i+a)%N][j][k] \
                                                         + Ising[i][(j+a)%N][k] \
                                                          + Ising[i][j][(k+a)%N] )
  else:   # non periodic boundary condition
    for i in range(N):
      for j in range(N):
        for k in range(N):
          pass
  return energy/2.0
```

```
def transition_energy(Ising,i,j,k):
  '''
  This function calculates the energy change of the state transition
  '''
  dE = 0
  for a in [-1,1]:
    dE += 2*J* Ising[i][j][k]* (Ising[(i+a)%N][j][k]+Ising[i][(j+a)%N][k]+Ising[i][j][(k+a)%N] )
  return dE


def transition(Ising,beta,initial_energy):
  '''
  This function performs state transition of the 3D Ising model and returns the new state and energy
  '''
  (i,j,k) = tuple(np.random.randint(0, N, size=3)) # random spin position
  dE = transition_energy(Ising,i,j,k)

  if (dE<=0 or np.random.random() < np.exp(-beta*dE)): # new state accepted
    Ising[i][j][k] = -1*Ising[i][j][k]
    return Ising,initial_energy+dE

  else : return Ising,initial_energy     # new state rejected


def Calc_Magnetization(Ising):
  '''
  This function calculates the magnetization of the 3D Ising model
  '''
  return np.sum(Ising)
```

Initialising the Spin systems at different temperatures

```
Ising = [np.ones((N,N,N)) for i in range(len(beta))] # 3D Ising models
Initial_energy = NNInteraction_energy(Ising[0])
```

Initiating the Simulation

Equilibriation

```
Iter = 20000  # number of iterations
Energy = np.zeros((len(beta),Iter+1))
for i in range(len(beta)): Energy[i][0] = Initial_energy  # setting initial energy of spin systems

# Stabilising Spin systems initiated at different Temperatures
for i in range(len(beta)):  # Spin system at T[i] temp

  for j in range(1,Iter+1): # propagating ith system through markov chain
    Ising[i],Energy[i][j] = transition(Ising[i],beta[i],Energy[i][j-1])

  plt.plot(Energy[i])
  plt.xlabel('Iterations')
  plt.ylabel('Energy')
  plt.title(f'Energy vs Iterations at T={T[i]}')
  plt.autoscale(enable=True, axis='y')
  plt.savefig(
     f'//content//drive//MyDrive//Colab Notebooks//pyl435_assgn1_plots//Stabilising_plots//T{T[i]}_iter{Iter}_N{N}.png'
      )
  plt.clf()
```

```
<Figure size 640x480 with 0 Axes>
```

Taking running sums of measurables of sequentially sampled states

```
Iter = 10000  # Number of states to be sampled
E_avg = np.zeros((len(beta),Iter+1)); M_avg = np.zeros((len(beta),Iter+1))
E_squared_avg = np.zeros((len(beta),Iter+1)); M_squared_avg = np.zeros( (len(beta),Iter+1) )
for i in range(len(beta)):
# Copying current energy and magnetization of the ith ising model to E_avg[i] and M_avg[i] lists
  E_avg[i][0] = Energy[i][-1]
  M_avg[i][0] = Calc_Magnetization(Ising[i])
  E_squared_avg[i][0] = Energy[i][-1]**2
  M_squared_avg[i][0] = M_avg[i][0]**2

for i in range(len(beta)): # Spin system at T[i] temp

  e = E_avg[i][0]; m = M_avg[i][0]
  e_sq = E_squared_avg[i][0]; m_sq = M_squared_avg[i][0]
  Cur_energy = Energy[i][-1]

  for j in range(1,Iter+1): # propagating ith system through markov chain
    Ising[i],Cur_energy = transition(Ising[i],beta[i],Cur_energy)
    mu = Calc_Magnetization(Ising[i])

    e+=Cur_energy; e_sq+=Cur_energy**2
    m+=mu; m_sq+=mu**2
    # Taking running average
    E_avg[i][j] = e/(j+1) ; E_squared_avg[i][j] = e_sq/(j+1)
    M_avg[i][j] = m/(j+1) ; M_squared_avg[i][j] = m_sq/(j+1)
```

plotting running average vs iterations of E, M, E_squared and M_squared

```
for i in range(len(beta)):  # Energy running average vs iterations
  plt.plot(E_avg[i])
  plt.xlabel('Iterations')
  plt.ylabel(' <Energy>')
  plt.title(f'<Energy> vs Iterations at T={T[i]}')
  plt.autoscale(enable=True, axis='y')
  plt.savefig(
      f'//content//drive//MyDrive//Colab Notebooks//pyl435_assgn1_plots//running_avg//E_T{T[i]}_iter{Iter}_N{N}.png'
      )
  plt.clf()
```

```
<Figure size 640x480 with 0 Axes>
```

```
for i in range(len(beta)):  # Energy_squared running average vs iterations
  plt.plot(E_squared_avg[i])
  plt.xlabel('Iterations')
  plt.ylabel(' <Energy^2>')
  plt.title(f'<Energy^2> vs Iterations at T={T[i]}')
  plt.autoscale(enable=True, axis='y')
  plt.savefig(
      f'//content//drive//MyDrive//Colab Notebooks//pyl435_assgn1_plots//running_avg//Esq_T{T[i]}_iter{Iter}_N{N}.png'
      )
  plt.clf()
```

```
<Figure size 640x480 with 0 Axes>
```

```
for i in range(len(beta)):  # M running average vs iterations
  plt.plot(M_avg[i])
  plt.xlabel('Iterations')
  plt.ylabel(' <Magnetization>')
  plt.title(f'<M> vs Iterations at T={T[i]}')
  plt.autoscale(enable=True, axis='y')
  plt.savefig(
      f'//content//drive//MyDrive//Colab Notebooks//pyl435_assgn1_plots//running_avg//M_T{T[i]}_iter{Iter}_N{N}.png'
      )
  plt.clf()
```

```
<Figure size 640x480 with 0 Axes>
```

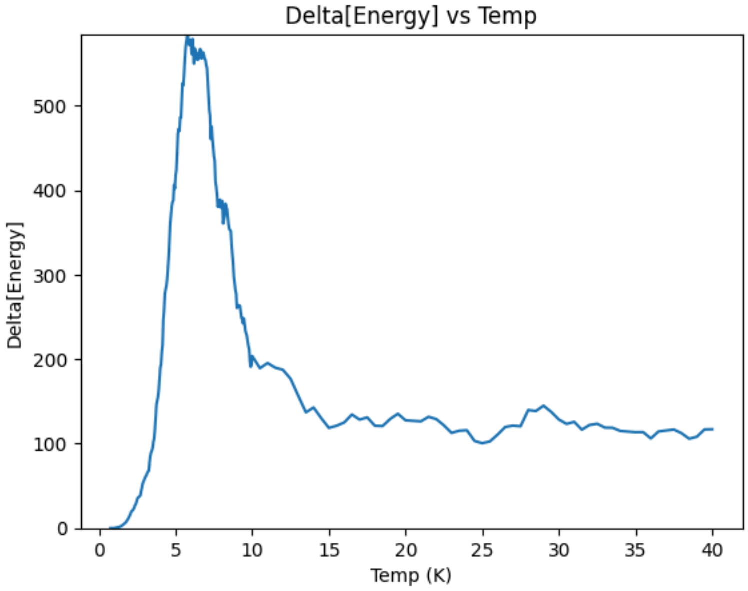
```
for i in range(len(beta)):  # M_squared running average vs iterations
  plt.plot(M_squared_avg[i])
  plt.xlabel('Iterations')
  plt.ylabel(' <M^2>')
  plt.title(f'<M^2> vs Iterations at T={T[i]}')
  plt.autoscale(enable=True, axis='y')
  plt.savefig(
      f'//content//drive//MyDrive//Colab Notebooks//pyl435_assgn1_plots//running_avg//Msq_T{T[i]}_iter{Iter}_N{N}.png'
      )
  plt.clf()
```

<Figure size 640x480 with 0 Axes>

Calculating variance of Energy and Magnetisation with temp

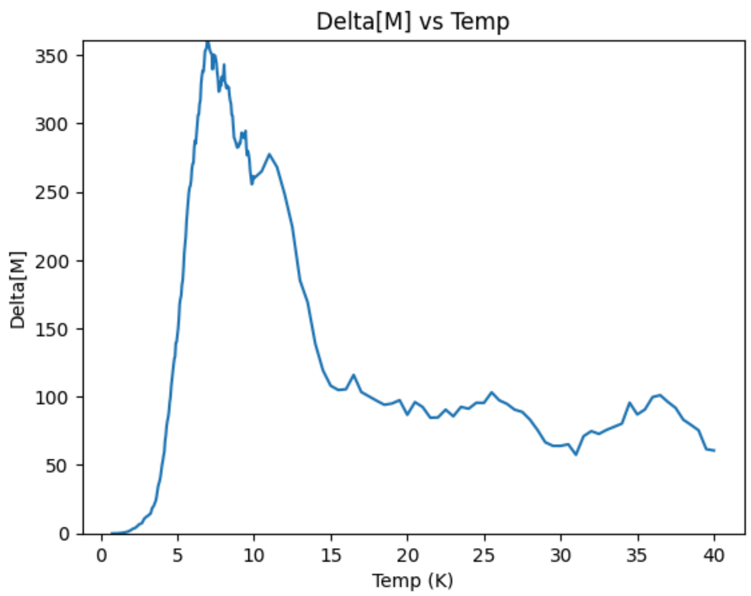
```
E_var = E_squared_avg.T[-1] - E_avg.T[-1]**2;E_std = np.sqrt(E_var)
M_var = M_squared_avg.T[-1] - M_avg.T[-1]**2;M_std = np.sqrt(M_var)
```

```
E_std_smooth = pd.concat([pd.Series(E_std[:200+3]).rolling(window=15).mean()[:200],
                      pd.Series(E_std[200-7:]).rolling(window=7).mean()[7:] ])
plt.plot(T,E_std_smooth)
plt.xlabel('Temp (K)')
plt.ylabel('Delta[Energy]')
plt.title(f'Delta[Energy] vs Temp')
plt.autoscale(enable=True, axis='y', tight=True)
plt.savefig(
      f'//content//drive//MyDrive//Colab Notebooks//pyl435_assgn1_plots//E_std_vs_T_till_T{T[-1]}_iter{Iter}.png'
      )
plt.show()
plt.clf()
```



<Figure size 640x480 with 0 Axes>

```
M_std_smooth = pd.concat([pd.Series(M_std[:200+3]).rolling(window=15).mean()[:200],
                      pd.Series(M_std[200-7:]).rolling(window=7).mean()[7:] ])
plt.plot(T,M_std_smooth)
plt.xlabel('Temp (K)')
plt.ylabel('Delta[M]')
plt.title(f'Delta[M] vs Temp')
plt.autoscale(enable=True, axis='y', tight=True)
plt.savefig(
      f'//content//drive//MyDrive//Colab Notebooks//pyl435_assgn1_plots//M_std_vs_T_till_T{T[-1]}_iter{Iter}.png'
      )
plt.show()
plt.clf()
```



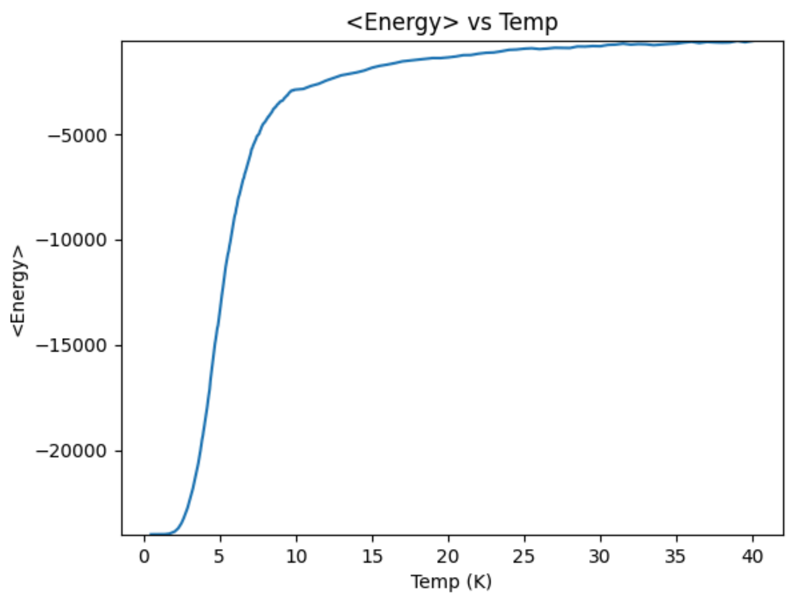
<Figure size 640x480 with 0 Axes>

Calculating Susceptibility and specific heat

```
C = (beta/T)*(E_squared_avg.T[-1] - E_avg.T[-1]**2)
Chi = (beta)*(M_squared_avg.T[-1] - M_avg.T[-1]**2)
```

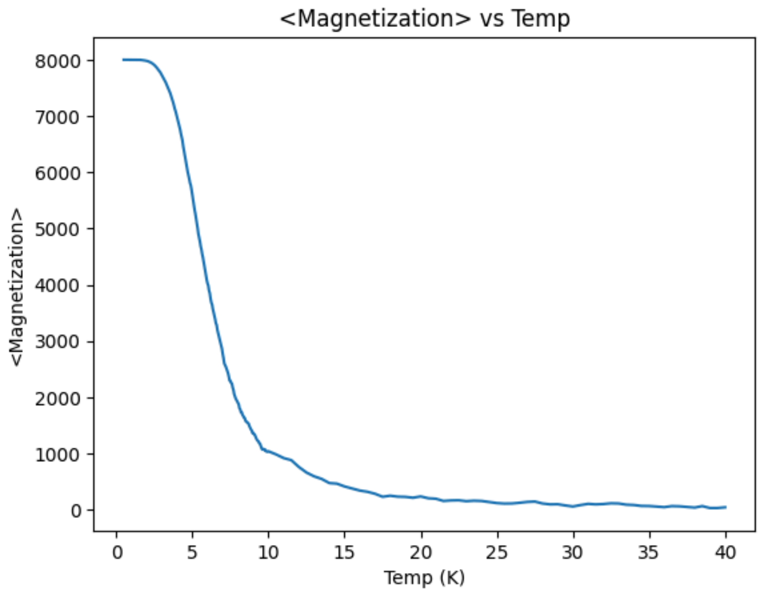
Plots

```
E_avg_smooth = pd.concat([pd.Series(E_avg.T[-1][:200+2]).rolling(window=10).mean()[:200],
                          pd.Series(E_avg.T[-1][200-5:]).rolling(window=5).mean()[5:] ])
plt.plot(T,E_avg_smooth)
plt.xlabel('Temp (K)')
plt.ylabel('<Energy>')
plt.title(f'<Energy> vs Temp')
plt.autoscale(enable=True, axis='y', tight=True)
plt.savefig(
      f'//content//drive//MyDrive//Colab Notebooks//pyl435_assgn1_plots//E_vs_T_till_T{T[-1]}_iter{Iter}.png'
      )
plt.show()
plt.clf()
```



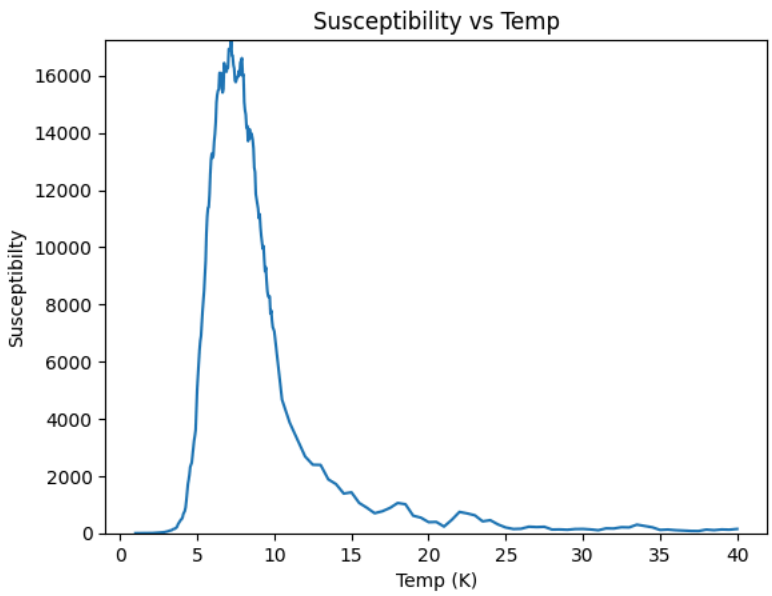
<Figure size 640x480 with 0 Axes>

```
M_avg_smooth = pd.concat([pd.Series(M_avg.T[-1][:200+2]).rolling(window=10).mean()[:200],
                      pd.Series(M_avg.T[-1][200-5:]).rolling(window=5).mean()[5:] ])
plt.plot(T,M_avg_smooth)
plt.xlabel('Temp (K)')
plt.ylabel('<Magnetization>')
plt.title(f'<Magnetization> vs Temp')
plt.autoscale(enable=True, axis='y')
plt.savefig(
      f'//content//drive//MyDrive//Colab Notebooks//pyl435_assgn1_plots//M_vs_T_till_T{T[-1]}_iter{Iter}.png'
      )
plt.show()
plt.clf()
```



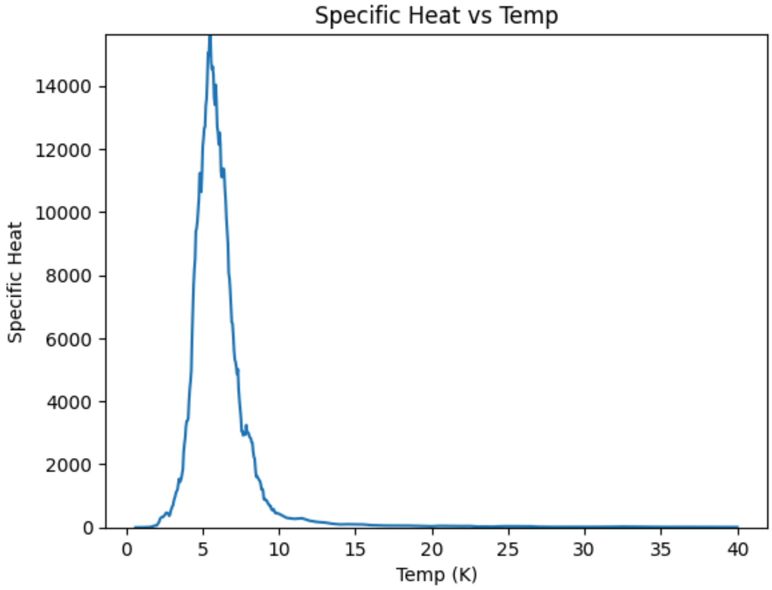

```
<Figure size 640x480 with 0 Axes>
```

```
Chi_smooth = pd.concat([pd.Series(Chi[:200+1]).rolling(window=20).mean()[:200],
                      pd.Series(Chi[200-10:]).rolling(window=3).mean()[10:] ])
plt.plot(T,Chi_smooth)
plt.xlabel('Temp (K)')
plt.ylabel('Susceptibilty')
plt.title(f'Susceptibility vs Temp')
plt.autoscale(enable=True, axis='y', tight=True)
plt.savefig(
      f'//content//drive//MyDrive//Colab Notebooks//pyl435_assgn1_plots//Chi_vs_T_till_T{T[-1]}_iter{Iter}.png'
      )
plt.show()
plt.clf()
```




```
<Figure size 640x480 with 0 Axes>
```

```
C_smooth = pd.concat([pd.Series(C[:200+2]).rolling(window=12).mean()[:200],
                      pd.Series(C[200-6:]).rolling(window=5).mean()[6:] ])
plt.plot(T,C_smooth)
plt.xlabel('Temp (K)')
plt.ylabel('Specific Heat')
plt.title(f'Specific Heat vs Temp')
plt.autoscale(enable=True, axis='y', tight=True)
plt.savefig(
      f'//content//drive//MyDrive//Colab Notebooks//pyl435_assgn1_plots//C_vs_T_till_T{T[-1]}_iter{Iter}.png'
      )
plt.show()
plt.clf()
```




```
<Figure size 640x480 with 0 Axes>
```