Methods

Here I have outlined the programs used and how to run them on the GPU machine available at TIFR.

1.1 Implementation

1.1.1 Ising Model

Monte Carlo Algorithm

To begin with study of samples generated from deep learning models, we require data to train them. This data is generated using the Wolff Cluster algorithm, for various lattice sizes. The following code implements it using parallel processing for faster implementation.
CODE

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

import os
from tqdm import tqdm
import multiprocessing as mp
from itertools import repeat

train_path = '../../data/Ising/train/'
valid_path = '../../data/Ising/valid/'

def makeArray(L):
    array = np.random.randint(0,2,size=(L,L), dtype=int)
    array[array==0] = -1
    return array

def cluster_flip(array,cluster):# Flip all spins in a cluster
    """
    Parameters:
        array: numpy array containing the current spin configuration
        cluster: List of spins to be flipped
    """
    for pair in cluster:
        i,j=pair
        array[i,j]*=-1
```

```python

def Wolff(array,T,calcE=False):
    # Wolff algorithm for a square lattice
    """
    Parameters:
        array: numpy array containing the current spin configuration
        T: Temperature
        calcE: Flag to calculate and return the average spin energy for the final configuration.
    """
    L,_ = array.shape
    cluster_i = np.zeros((L,L)) # spins already in cluster

    i,j = np.random.randint(0,L,size=2)
    spin = array[i,j]
    stack = [(i,j)]
    cluster_i[i,j]=1

    cluster = [(i,j)]
    while len(stack)>0:
        i,j = stack.pop()
        neighbors = [(i,(j+1)%L),(i,(j-1)%L),((i+1)%L,j),((i-1)%L,j)]
        for pair in neighbors:
            l,m = pair
            if (array[l,m]==spin and cluster_i[l,m]==0 and np.random.random()< (1.0-np.exp(-2.0/T))):
                cluster.append((l,m))
                stack.append((l,m))
                cluster_i[l,m]=1

    cluster_flip(array,cluster)

    if calcE:
        avgE=0.0
        for i in range(L):
            for j in range(L):
                spin_final = array[i,j]
                neighbor_sum = array[(i+1)%L,j]+array[(i-1)%L,j]+array[i,(j+1)%L]+array[i,(j-1)%L]
                E_final = -spin_final*neighbor_sum
                avgE+=float(E_final)

        avgE/=float(L**2)
        return avgE

def sim(array,T,L,steps,step_r,lattice):
    #MC simulation for a square lattice, using the Wolff algorithm
    for step in range(steps):
        Wolff(array,T)
        if step>step_r and step%divide==0:
          i=int((step-step_r)/divide)
          lattice[i]=array

def save_data(T,L,steps,step_r,lattice):
  array=makeArray(L)
  sim(array,T,L,steps,step_r,lattice)
  lattice=np.reshape(lattice,(length,L*L))
```

```
    np.savetxt("isingdata_"+str(L*L)+f"{T:.2f}"+".csv",lattice,delimiter=",")

Temps=np.arange(1.5,3.2,0.1) #in Kelvin

L=25
step_r=200 #configurations to be removed
steps = 200000+step_r
divide=20 #autocorrelation tim
length=int((steps-step_r)/divide) #size of dataset
lattice=np.zeros((len(Temps),length,L,L))
#Parallel processing
jobs=zip(Temps,repeat(L),repeat(steps),repeat(step_r),lattice)
p = mp.Pool(5)
p.starmap(save_data, jobs)
p.close()
p.join()
```

Listing 1.1: isingdata25_wolff.py

This program saves the data as "isingdata_L^2T{2 digits}.csv" with L^2 rows and $steps - step\_r$ columns. The `mp.Pool()` function is used to implement parallel processing.

To check is the data is generated correctly OR to study the properties of Ising model for various lattice sizes and temperatures, we can plot certain parameters. Here we plot the magnetisation defined as $M = \frac{1}{N} \times \Sigma_N \{\frac{1}{L^2} \times |\Sigma_{L^2} s_i|\} = \frac{1}{N} \times \Sigma_N s$, where s_i is the spin at i^{th} lattice position, and $N = steps - step\_r$.

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

L=35
Size=3000

mag=np.zeros((5))

sample = np.genfromtxt('/user1/utkarsh/dataising350', delimiter = ',') #changing the file type to .csv format
np.savetxt("dataising350.csv",sample, delimiter = ',')
print(sample)
df2 = pd.read_csv('/user1/utkarsh/dataising350.csv') #reading the .csv file
data=df2.to_numpy()
np.savetxt('mag.csv',data)
print(data.shape)
m=np.sum(data,axis=1)
m=np.sum(m)
print(m)
mag[0]=(m/L**2)/Size

sample = np.genfromtxt('/user1/utkarsh/dataising352', delimiter = ',')
np.savetxt("dataising352.csv",sample, delimiter = ',')
df2 = pd.read_csv('/user1/utkarsh/dataising352.csv')
data=df2.to_numpy()
m=np.sum(data,axis=1)
m=np.sum(m)
mag[1]=(m/L**2)/Size

```

```
sample = np.genfromtxt('/user1/utkarsh/dataising354', delimiter = ',')
np.savetxt("dataising354.csv",sample, delimiter = ',')
df2 = pd.read_csv('/user1/utkarsh/dataising354.csv')
data=df2.to_numpy()
m=np.sum(data,axis=1)
m=np.sum(m)
mag[2]=(m/L**2)/Size

sample = np.genfromtxt('/user1/utkarsh/dataising356', delimiter = ',')
np.savetxt("dataising356.csv",sample, delimiter = ',')
df2 = pd.read_csv('/user1/utkarsh/dataising356.csv')
data=df2.to_numpy()
m=np.sum(data,axis=1)
m=np.sum(m)
mag[3]=(m/L**2)/Size

sample = np.genfromtxt('/user1/utkarsh/dataising358', delimiter = ',')
np.savetxt("dataising358.csv",sample, delimiter = ',')
df2 = pd.read_csv('/user1/utkarsh/dataising358.csv')
data=df2.to_numpy()
m=np.sum(data,axis=1)
m=np.sum(m)
mag[4]=(m/L**2)/Size

print(mag)
plt.plot(mag)
plt.savefig('mag350_mc.png')
```

Listing 1.2: magplot_mc_35.py

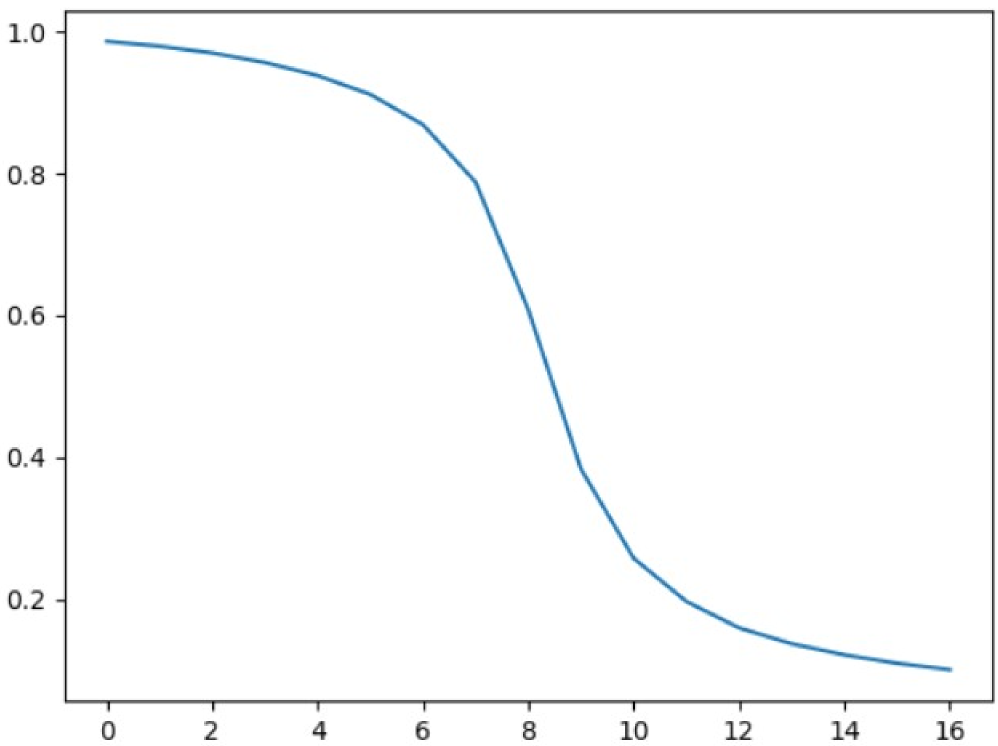

Figure 1.1: Magnetisation v/s Temperature index

The second parameter of interest is the Binder Cumulant given as $U = 1 - \frac{<s^4>}{3<s^2>^2}$.

```

import pandas as pd
from matplotlib import pyplot as plt
import numpy as np

nbeta=5 # Number of data points
time=3000 #Number of samples per data point
L=35 #Lattice size

data=np.zeros((nbeta,time,d**2))
df2 = pd.read_csv('/user1/utkarsh/rbmdata350.csv',header=None) #keep
    the required file name
data[0]=df2.to_numpy()
df2 = pd.read_csv('/user1/utkarsh/rbmdata352.csv',header=None)
data[1]=df2.to_numpy()
df2 = pd.read_csv('/user1/utkarsh/rbmdata354.csv',header=None)
data[2]=df2.to_numpy()
df2 = pd.read_csv('/user1/utkarsh/rbmdata356.csv',header=None)
data[3]=df2.to_numpy()
df2 = pd.read_csv('/user1/utkarsh/rbmdata358.csv',header=None)
data[4]=df2.to_numpy()

mag_4_avg=np.zeros(nbeta) #Mag**4 average
mag_sq_avg=np.zeros(nbeta) #Mag**2 average
betas = np.linspace(0.43,0.45,5) #beta values


for i in range(nbeta): #Loop over all data points
  mag=np.sum(data[i],axis=1)

  for o in range(time): #Calculation for each data point
    mag_4_avg[i]+=mag[o]*mag[o]*mag[o]*mag[o]
    mag_sq_avg[i]+=mag[o]*mag[o]
  print("sweep",i)
  mag_4_avg[i]=mag_4_avg[i]/time
  mag_sq_avg[i]=mag_sq_avg[i]/time
U1=1-(1/3)*(mag_4_avg/(mag_sq_avg*mag_sq_avg))


nbeta=5
time=3000
L=25

data=np.zeros((nbeta,time,d**2))
df2 = pd.read_csv('/user1/utkarsh/rbmdata250_.csv',header=None)
data[0]=df2.to_numpy()
df2 = pd.read_csv('/user1/utkarsh/rbmdata252_.csv',header=None)
data[1]=df2.to_numpy()
df2 = pd.read_csv('/user1/utkarsh/rbmdata254.csv',header=None)
data[2]=df2.to_numpy()
df2 = pd.read_csv('/user1/utkarsh/rbmdata256.csv',header=None)
data[3]=df2.to_numpy()
df2 = pd.read_csv('/user1/utkarsh/rbmdata258.csv',header=None)
data[4]=df2.to_numpy()

mag_4_avg=np.zeros(nbeta)
```

```
mag_sq_avg=np.zeros(nbeta)
betas = np.linspace(0.43,0.45,5)


for i in range(nbeta):
  mag=np.sum(data[i],axis=1)

  for o in range(time):
    mag_4_avg[i]+=mag[o]*mag[o]*mag[o]*mag[o]
    mag_sq_avg[i]+=mag[o]*mag[o]
  print("sweep",i)
  mag_4_avg[i]=mag_4_avg[i]/time
  mag_sq_avg[i]=mag_sq_avg[i]/time
U2=1-(1/3)*(mag_4_avg/(mag_sq_avg*mag_sq_avg))

plt.plot(betas,U1/U2)
#plt.plot(betas,U2)
plt.savefig('UIratio.png')
```

Listing 1.3: BinderCumulant_Ratio25_35.py

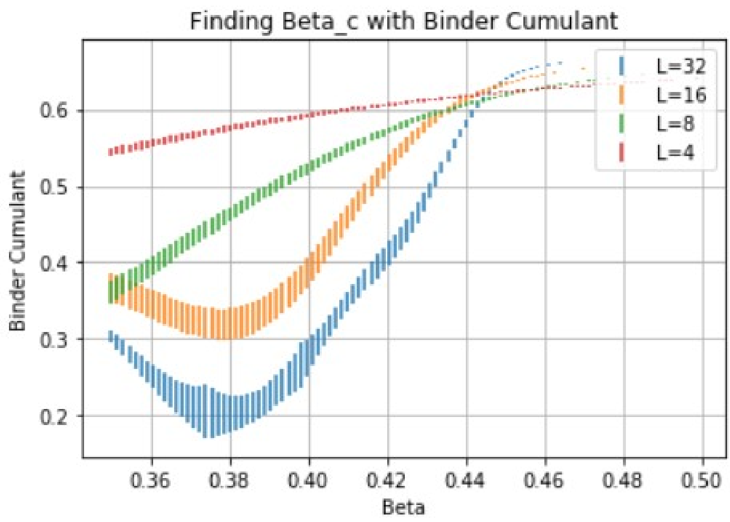

Figure 1.2: Binder Cumulant

Another parameter is χ i.e., the susceptibility given by $\chi = < \frac{s^2}{L^2} >$.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
L=35 #Lattice size
mag=np.zeros((5))
chi=np.zeros((5))
sample = np.genfromtxt('/user1/utkarsh/dataising350', delimiter = ',')
    #350 = lattice size=35, temperature index value = 0
np.savetxt("dataising350.csv",sample, delimiter = ',')
print(sample)
df2 = pd.read_csv('/user1/utkarsh/dataising350.csv')
data=df2.to_numpy()
np.savetxt('mag.csv',data)
print(data.shape)
```

```
m=np.sum(data,axis=1)
msq=m*m
m=np.sum(m)
msq=np.sum(msq)
print(m)
print(msq)
mag[0]=(m/L**2)/3000
chi[0]=(msq/L**4)/3000
sample = np.genfromtxt('/user1/utkarsh/dataising352', delimiter = ',')
np.savetxt("dataising352.csv",sample, delimiter = ',')
df2 = pd.read_csv('/user1/utkarsh/dataising352.csv')
data=df2.to_numpy()
m=np.sum(data,axis=1)
msq=m*m
m=np.sum(m)
msq=np.sum(msq)
print(m)
print(msq)
mag[1]=(m/L**2)/3000
chi[1]=(msq/L**4)/3000
sample = np.genfromtxt('/user1/utkarsh/dataising354', delimiter = ',')
np.savetxt("dataising354.csv",sample, delimiter = ',')
df2 = pd.read_csv('/user1/utkarsh/dataising354.csv')
data=df2.to_numpy()
m=np.sum(data,axis=1)
msq=m*m
m=np.sum(m)
msq=np.sum(msq)
print(m)
print(msq)
mag[2]=(m/L**2)/3000
chi[2]=(msq/L**4)/3000
sample = np.genfromtxt('/user1/utkarsh/dataising356', delimiter = ',')
np.savetxt("dataising356.csv",sample, delimiter = ',')
df2 = pd.read_csv('/user1/utkarsh/dataising356.csv')
data=df2.to_numpy()

m=np.sum(data,axis=1)
msq=m*m
m=np.sum(m)
msq=np.sum(msq)
print(m)
print(msq)
mag[3]=(m/L**2)/3000
chi[3]=(msq/L**4)/3000
sample = np.genfromtxt('/user1/utkarsh/dataising358', delimiter = ',')
np.savetxt("dataising358.csv",sample, delimiter = ',')
df2 = pd.read_csv('/user1/utkarsh/dataising358.csv')
data=df2.to_numpy()

m=np.sum(data,axis=1)
msq=m*m
m=np.sum(m)
msq=np.sum(msq)
print(m)
print(msq)
mag[4]=(m/L**2)/3000
chi[4]=(msq/L**4)/3000
```

```
print(mag)
print(chi)
plt.plot(chi)
plt.savefig('chi350_mc.png')
```

Listing 1.4: chi_plot_mc.py

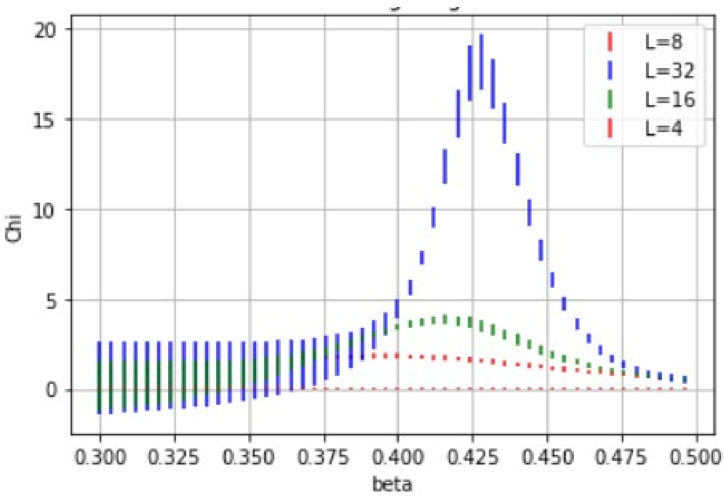

Figure 1.3: $\chi\, v/s\, \beta$

Restricted Boltzmann Machine

A Restricted Boltzmann Machine is trained with dataset generated from the Wolff Cluster Algorithm. We had flatten the dataset to a 2D matrix of shape $[N, L^2]$ and feed it in the input nodes. The second layer size is a hyperparamter. The following program implements the training for one value of the temperature. The complete code can be found here.

```
import numpy as np
import random
import numpy as np
import math
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import os

epoch=25
size_h=1500
#return the average of h given x
def h_mean(x,w,b):
    #x: visible layer. One row is one data
    #w: weight, size_h*784
    #b: bias, size_h

    #return data*size_h
    b_shaped=np.reshape(b,(1,b.size))
    b_shaped=np.repeat(b_shaped,x.shape[0],axis=0)
    y=np.dot(x,w.T)+b_shaped
    return 1.0/(np.exp(-y)+1.0)
```

```

def x_mean(h,w,c):
    #h: visible layer. One row is one data
    #w: weight, size_h*784
    #c: bias, size_h

    c_shaped=np.reshape(c,(1,c.size))
    c_shaped=np.repeat(c_shaped,h.shape[0],axis=0)
    y=np.dot(h,w)+c_shaped
    return np.tanh(y),1/(np.exp(-1.0*y)+1.0)

def gibbs_cd(x,kcd,w,b,c):
    h_p=h_mean(x,w,b)
    rad_h=np.random.rand(h_p.shape[0],h_p.shape[1])
    h_samp=np.zeros(h_p.shape)
    h_samp[rad_h<h_p]=1
    for i in range(kcd):
        #sample x
        _, x_prob=x_mean(h_samp,w,c)
        rad_x=np.random.rand(x_prob.shape[0],x_prob.shape[1])
        x_samp=-np.ones(x_prob.shape)
        x_samp[rad_x<x_prob]=1
        #sample h
        h_p=h_mean(x_samp,w,b)
        rad_h=np.random.rand(h_p.shape[0],h_p.shape[1])
        h_samp=np.zeros(h_p.shape)
        h_samp[rad_h<h_p]=1
    return x_samp

def loss(x_in,x_p):
    l=(x_in-x_p)**2
    loss=np.sum(l)/x_in.size
    return loss

###################################################################

df2 = pd.read_csv('/user1/utkarsh/isingdata_6251.50.csv')
data=df2.to_numpy()

trX=data[0:8000]
test_data=data[8000:10000]

spin_flatten=trX

#learning rate
gamma=0.005
#momentum
beta=0.0
#l1 regularization
lambd=0

#number of hidden units

size_v=spin_flatten.shape[1]
w=np.reshape(np.random.normal(scale=0.1,size=size_v*size_h),(size_h,
    size_v))
b=np.zeros(size_h)
c=np.zeros(size_v)
```

```

batch=100 #batch size
kcd=20    #cd number


train_loss=np.zeros(epoch)
grad_w0=np.zeros(w.shape)
grad_b0=np.zeros(b.shape)
grad_c0=np.zeros(c.shape)

print('start')

for i in range(epoch):
    spin_train=np.random.permutation(spin_flatten)
    for j in range(math.floor(spin_train.shape[0]/batch)):
        x_in=spin_train[j*batch:j*batch+batch,:]

        x_sampling= gibbs_cd(x_in,kcd,w,b,c)
        #gradient update
        grad_w=np.zeros(w.shape)
        for k in range(batch):
            x_k=np.reshape(x_in[k,:],(1,x_in[k,:].size))
            x_ksample=np.reshape(x_sampling[k,:],(1,x_sampling[k,:].size))
            grad_w=grad_w+(np.outer( h_mean(x_k,w,b),x_k)-np.outer(h_mean(x_ksample,w,b),x_ksample))


        w=w+gamma*(grad_w-lambd*np.sign(w))
        b=b+gamma*np.sum( h_mean(x_in,w,b)- h_mean(x_sampling,w,b),axis=0)


        c=c+gamma*np.sum(x_in-x_sampling,axis=0)

    print('epoch'+str(i))

    #training reconstruction
    h_p= h_mean(spin_train,w,b)
    x_p,_= x_mean(h_p,w,c)
    train_loss[i]= loss(spin_train,x_p)
    print('train'+str(train_loss[i]))


fnamew='w_ising'+'6251.50.csv'
np.savetxt(fnamew,w, delimiter=",")
fnameb='b_ising'+'6251.50.csv'
np.savetxt(fnameb,b, delimiter=",")
fnamec='c_ising'+'6251.50.csv'
np.savetxt(fnamec,c, delimiter=",")
```

Listing 1.5: rbmising25.py

Once the model is trained with the data for a given lattice size and temperature, we can generate data from the models with the following code.

```
import pandas as pd

#return the average of h given x
```

```
def h_mean(x,w,b):
    #x: visible layer. One row is one data
    #w: weight, size_h*784
    #b: bias, size_h

    #return data*size_h
    b_shaped=np.reshape(b,(1,b.size))
    b_shaped=np.repeat(b_shaped,x.shape[0],axis=0)
    y=np.dot(x,w.T)+b_shaped
    return 1.0/(np.exp(-y)+1.0)

def x_mean(h,w,c):
    #h: visible layer. One row is one data
    #w: weight, size_h*784
    #c: bias, size_h

    c_shaped=np.reshape(c,(1,c.size))
    c_shaped=np.repeat(c_shaped,h.shape[0],axis=0)
    y=np.dot(h,w)+c_shaped
    return np.tanh(y),1/(np.exp(-1.0*y)+1.0)

def gibbs_cd(x,kcd,w,b,c):
    h_p=h_mean(x,w,b)
    rad_h=np.random.rand(h_p.shape[0],h_p.shape[1])
    h_samp=np.zeros(h_p.shape)
    h_samp[rad_h<h_p]=1
    for i in range(kcd):
        #sample x
        _, x_prob=x_mean(h_samp,w,c)
        rad_x=np.random.rand(x_prob.shape[0],x_prob.shape[1])
        x_samp=-np.ones(x_prob.shape)
        x_samp[rad_x<x_prob]=1
        #sample h
        h_p=h_mean(x_samp,w,b)
        rad_h=np.random.rand(h_p.shape[0],h_p.shape[1])
        h_samp=np.zeros(h_p.shape)
        h_samp[rad_h<h_p]=1
    return x_samp

def loss(x_in,x_p):
    l=(x_in-x_p)**2
    loss=np.sum(l)/x_in.size
    return loss

#sampling from the rbm
import numpy as np
import math
import matplotlib
import matplotlib.pyplot as plt
#import rbm_xy_fun as rbm_fun
import os
def generate(n,w,b,c,x_data):

    for i in range(n):
        rad_x=np.random.rand(1,c.size)
        x_sampling=gibbs_cd(rad_x,kcd,w,b,c)
        h_p=h_mean(x_sampling,w,b)
        x_p,x_dat=x_mean(h_p,w,c)
```

```
        rad_x=np.random.rand(x_dat.shape[0],x_dat.shape[1])
        x_samp=-np.ones(x_dat.shape)
        x_samp[rad_x<x_p]=1
        x_shaped=np.reshape(x_samp,(d,d))
        x_data[i]=x_shaped

d=16
n=1000
kcd=100
sigma=1
m=np.zeros(17)
msq=np.zeros(17)
#
    ################################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2561.50.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2561.50.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2561.50.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[0]=np.sum(ms)
m[0]=np.sum(abs(mag))

np.savetxt("rbmdata2561.5.csv", x_data, delimiter=",")
#
    ################################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2561.60.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2561.60.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2561.60.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[1]=np.sum(ms)
m[1]=np.sum(abs(mag))

np.savetxt("rbmdata2561.6.csv", x_data, delimiter=",")
#
    ################################################################################
```

```

df = pd.read_csv('/user1/utkarsh/b_ising2561.70.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2561.70.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2561.70.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[2]=np.sum(ms)
m[2]=np.sum(abs(mag))

np.savetxt("rbmdata2561.7.csv", x_data, delimiter=",")
#
    ################################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2561.80.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2561.80.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2561.80.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[3]=np.sum(ms)
m[3]=np.sum(abs(mag))

np.savetxt("rbmdata2561.8.csv", x_data, delimiter=",")
#
    ################################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2561.90.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2561.90.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2561.90.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

```

```
mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[4]=np.sum(ms)
m[4]=np.sum(abs(mag))

np.savetxt("rbmdata2561.9.csv", x_data, delimiter=",")
#
    ################################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2562.00.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2562.00.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2562.00.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[5]=np.sum(ms)
m[5]=np.sum(abs(mag))

np.savetxt("rbmdata2562.0.csv", x_data, delimiter=",")
#
    ################################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2562.10.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2562.10.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2562.10.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[6]=np.sum(ms)
m[6]=np.sum(abs(mag))

np.savetxt("rbmdata2562.1.csv", x_data, delimiter=",")
#
    ################################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2562.20.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2562.20.csv', header=None)
c=df.to_numpy()
```

```
df = pd.read_csv('/user1/utkarsh/w_ising2562.20.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[7]=np.sum(ms)
m[7]=np.sum(abs(mag))

np.savetxt("rbmdata2562.2.csv", x_data, delimiter=",")
# ###############################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2562.30.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2562.30.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2562.30.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[8]=np.sum(ms)
m[8]=np.sum(abs(mag))

np.savetxt("rbmdata2562.3.csv", x_data, delimiter=",")
# ###############################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2562.40.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2562.40.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2562.40.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[9]=np.sum(ms)
m[9]=np.sum(abs(mag))

np.savetxt("rbmdata2562.4.csv", x_data, delimiter=",")
```

```
# ################################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2562.50.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2562.50.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2562.50.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[10]=np.sum(ms)
m[10]=np.sum(abs(mag))

np.savetxt("rbmdata2562.5.csv", x_data, delimiter=",")
# ################################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2562.60.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2562.60.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2562.60.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[11]=np.sum(ms)
m[11]=np.sum(abs(mag))

np.savetxt("rbmdata2562.6.csv", x_data, delimiter=",")
# ################################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2562.70.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2562.70.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2562.70.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

```

```
x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[12]=np.sum(ms)
m[12]=np.sum(abs(mag))

np.savetxt("rbmdata2562.7.csv", x_data, delimiter=",")
# ###############################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2562.80.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2562.80.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2562.80.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[13]=np.sum(ms)
m[13]=np.sum(abs(mag))

np.savetxt("rbmdata2562.8.csv", x_data, delimiter=",")
# ###############################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2562.90.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2562.90.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2562.90.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[14]=np.sum(ms)
m[14]=np.sum(abs(mag))

np.savetxt("rbmdata2562.9.csv", x_data, delimiter=",")
# ###############################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2563.00.csv', header=None)
b=df.to_numpy()
```

```python
df = pd.read_csv('/user1/utkarsh/c_ising2563.00.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2563.00.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[15]=np.sum(ms)
m[15]=np.sum(abs(mag))

np.savetxt("rbmdata2563.0.csv", x_data, delimiter=",")
#
    ##########################################################################################

df = pd.read_csv('/user1/utkarsh/b_ising2563.10.csv', header=None)
b=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/c_ising2563.10.csv', header=None)
c=df.to_numpy()
df = pd.read_csv('/user1/utkarsh/w_ising2563.10.csv', header=None)
w=df.to_numpy()

x_data=np.zeros((n,d,d))
generate(n,w,b,c,x_data)

x_data=np.reshape(x_data,(n,d**2))

mag=np.sum(x_data, axis=1)
ms=mag*mag
msq[16]=np.sum(ms)
m[16]=np.sum(abs(mag))


np.savetxt("rbmdata2563.1.csv", x_data, delimiter=",")

#
    ##########################################################################################

m=m/d**2
m=m/n

chi=(msq/d**4)/n

err=np.sqrt(abs(chi-m**2))
x=np.arange(1.5,3.2,0.1)

plt.errorbar(x,m,err)

plt.savefig("magrbm16_err_new.png")
```

Listing 1.6: rbmgenerate_err.py

The generated datafile is now used to reproduce the plots for magnetisation, magnetic susceptibility and Binder Cumulant. The following plots present these results.

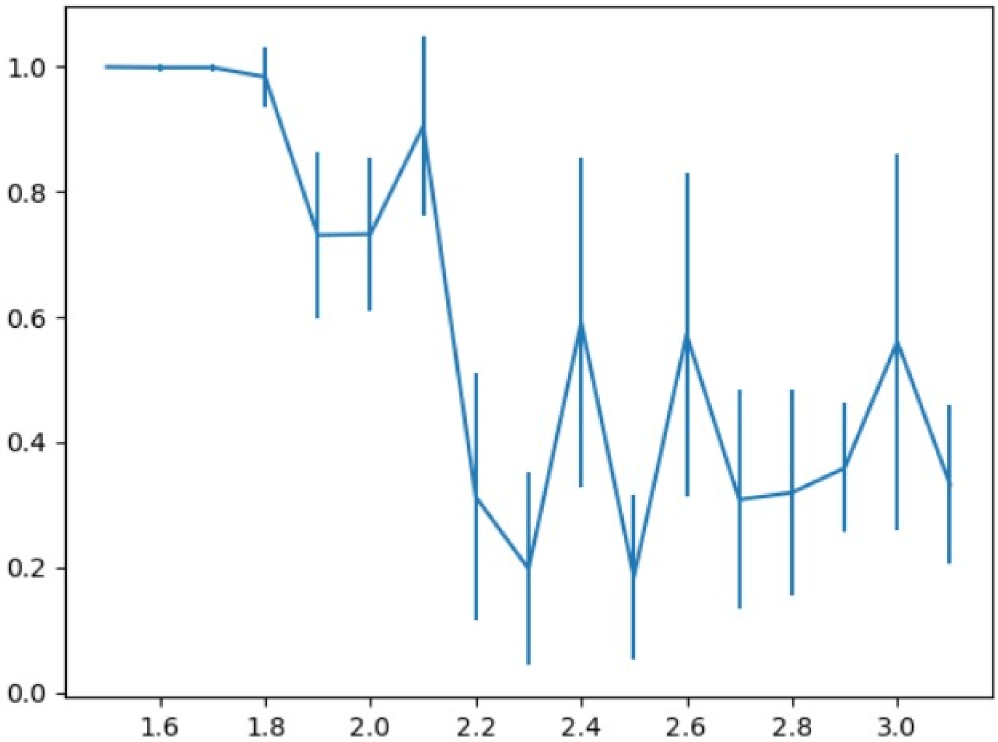

Figure 1.4: $Mv/sT(K)$, L=8

GPU Programs

1.1.2 XY Model

Monte Carlo Algorithm

Here is the code for the Monte Carlo algorithm to generate uncorrelated samples for the classical XY model.

```
# -*- coding: utf-8 -*-
"""dataxy32.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1
   IE1vmGDRo25BgQcJ2zu0SWi6ER1B6a-h
"""

import matplotlib
matplotlib.use('Agg')
import numpy as np
from numpy import linalg as LA
import matplotlib.pyplot as plt

L = 32
step_r = 3000
```

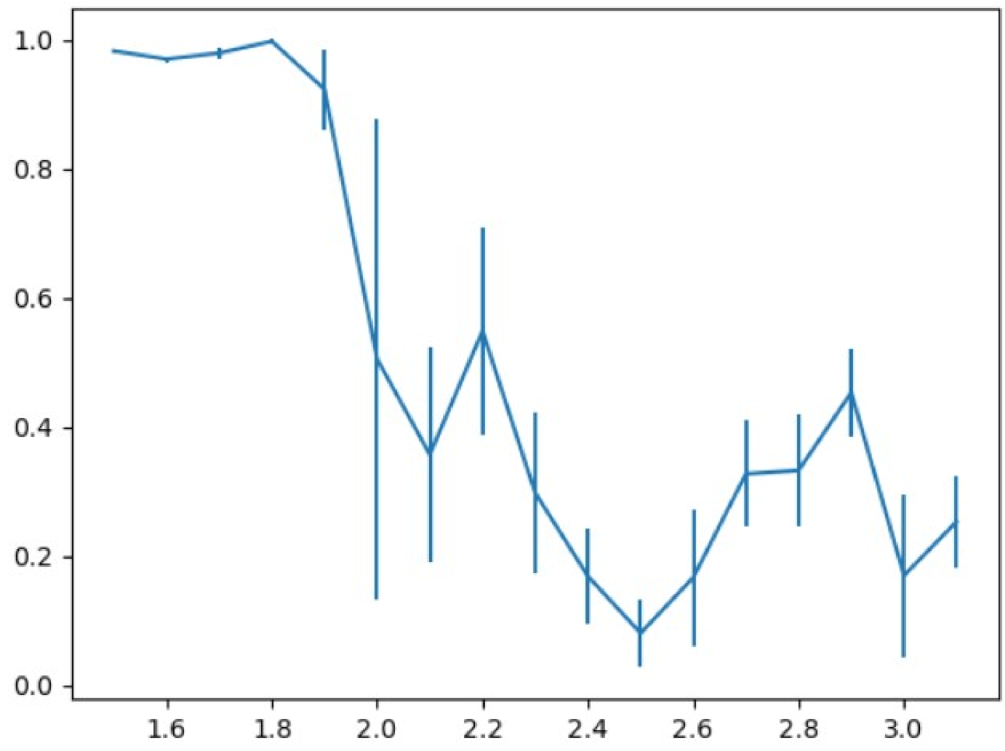

Figure 1.5: $Mv/sT(K)$, L=16

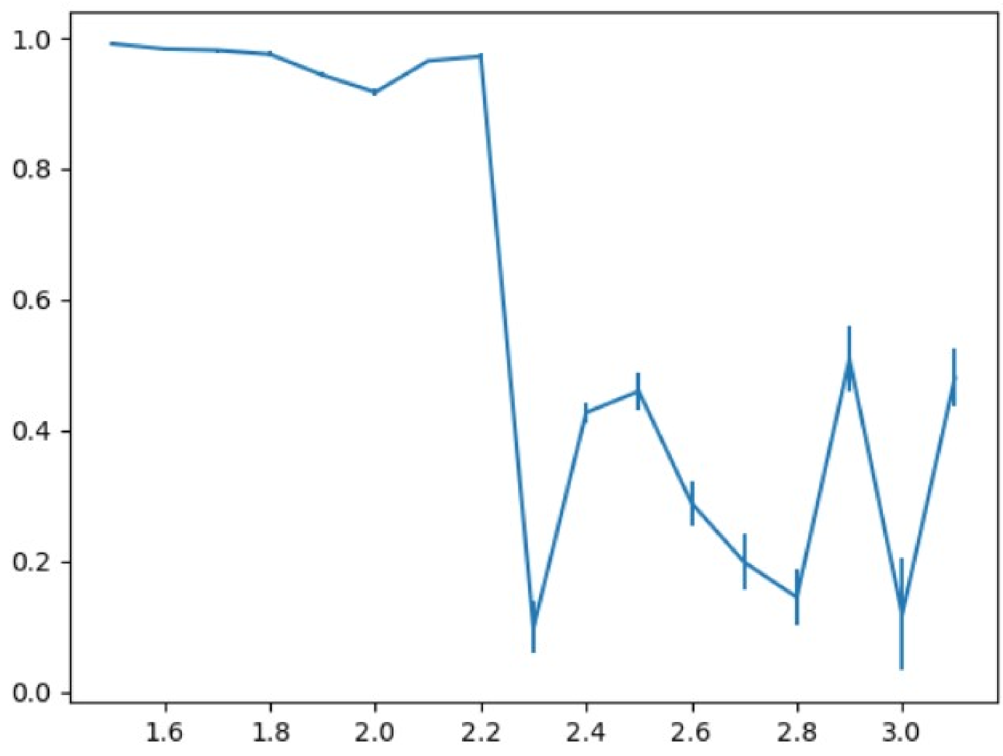

Figure 1.6: $Mv/sT(K)$, L=25

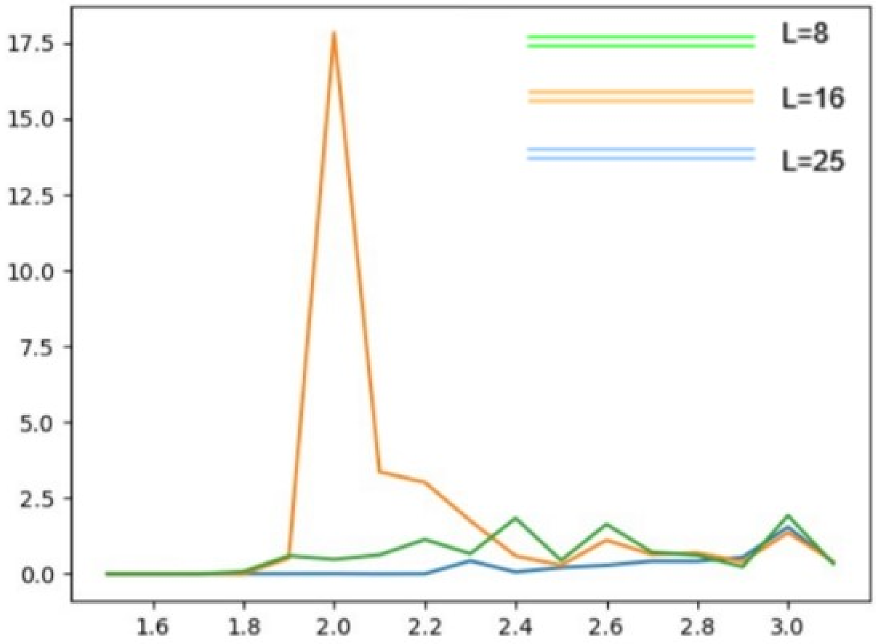

Figure 1.7: $\chi v/sT(K)$

```
step = 750000
nt=10
n_remove=30
J = 1 # J>0 to make it ferromagnetic


# Intitialize the XY network
def init():
    return np.random.rand(L, L)*2*np.pi
    #return np.ones([L, L])

# periodic neighbor
def next(x):
    if x == L-1:
        return 0
    else:
        return x+1

# construct the bond lattice
def FreezeBonds(Ising,T,S):
    iBondFrozen = np.zeros([L,L])
    jBondFrozen = np.zeros([L,L])
    for i in np.arange(L):
        for j in np.arange(L):
            freezProb_nexti = 1 - np.exp(-2 * J * S[i][j] * S[next(i)][
    j] / T)
            freezProb_nextj = 1 - np.exp(-2 * J * S[i][j] * S[i][next(j
    )] / T)
            if (Ising[i][j] == Ising[next(i)][j]) and (np.random.rand()
     < freezProb_nexti):
                iBondFrozen[i][j] = 1
```

```
            if (Ising[i][j] == Ising[i][next(j)]) and (np.random.rand()
    < freezProb_nextj):
                jBondFrozen[i][j] = 1
    return iBondFrozen, jBondFrozen

# H-K algorithm to identify clusters
def properlabel(prp_label,i):
    while prp_label[int(i)] != i:
        i = prp_label[int(i)]
    return i

# Swendsen-Wang cluster
def clusterfind(iBondFrozen,jBondFrozen):
    cluster = np.zeros([L, L])
    prp_label = np.zeros(L**2)
    label = 0
    for i in np.arange(L):
        for j in np.arange(L):
            bonds = 0
            ibonds = np.zeros(4)
            jbonds = np.zeros(4)

            # check to (i-1,j)
            if (i > 0) and iBondFrozen[i-1][j]:
                ibonds[bonds] = i-1
                jbonds[bonds] = j
                bonds += 1
            # (i,j) at i edge, check to (i+1,j)
            if (i == L-1) and iBondFrozen[i][j]:
                ibonds[bonds] = 0
                jbonds[bonds] = j
                bonds += 1
            # check to (i,j-1)
            if (j > 0) and jBondFrozen[i][j-1]:
                ibonds[bonds] = i
                jbonds[bonds] = j-1
                bonds += 1
            # (i,j) at j edge, check to (i,j+1)
            if (j == L-1) and jBondFrozen[i][j]:
                ibonds[bonds] = i
                jbonds[bonds] = 0
                bonds += 1

            # check and label clusters
            if bonds == 0:
                cluster[i][j] = label
                prp_label[label] = label
                label += 1
            else:
                minlabel = label
                for b in np.arange(bonds):
                    plabel = properlabel(prp_label,cluster[int(ibonds[b
  ])][int(jbonds[b])])
                    if minlabel > plabel:
                        minlabel = plabel

                cluster[i][j] = minlabel
                # link to the previous labels
```

```
                for b in np.arange(bonds):
                    plabel_n = cluster[int(ibonds[b])][int(jbonds[b])]
                    prp_label[int(plabel_n)] = minlabel
                    # re-set the labels on connected sites
                    cluster[int(ibonds[b])][int(jbonds[b])] = minlabel
    return cluster, prp_label

# flip the cluster spins
def flipCluster(Ising,cluster,prp_label):
    for i in np.arange(L):
        for j in np.arange(L):
            # relabel all the cluster labels with the right ones
            cluster[i][j] = properlabel(prp_label,cluster[i][j])
    sNewChosen = np.zeros(L**2)
    sNew = np.zeros(L**2)
    flips = 0 # get the number of flipped spins to calculate the Endiff
    and Magdiff
    for i in np.arange(L):
        for j in np.arange(L):
            label = cluster[i][j]
            randn = np.random.rand()
            # mark the flipped label, use this label to flip all the
    cluster elements with this label
            if (not sNewChosen[int(label)]) and randn < 0.5:
                sNew[int(label)] = +1
                sNewChosen[int(label)] = True
            elif (not sNewChosen[int(label)]) and randn >= 0.5:
                sNew[int(label)] = -1
                sNewChosen[int(label)] = True

            if Ising[i][j] != sNew[int(label)]:
                Ising[i][j] = sNew[int(label)]
                flips += 1

    return Ising,flips

# Swendsen-Wang Algorithm in Ising model (with coupling constant
    dependency on sites)
# One-step for Ising
def oneMCstepIsing(Ising, S):
    [iBondFrozen, jBondFrozen] = FreezeBonds(Ising, T, S)
    [SWcluster, prp_label] = clusterfind(iBondFrozen, jBondFrozen)
    [Ising, flips] = flipCluster(Ising, SWcluster, prp_label)
    return Ising

# Decompose XY network to two Ising networks with project direction
    proj
def decompose(XY,proj):
    x = np.cos(XY)
    y = np.sin(XY)
    x_rot = np.multiply(x,np.cos(proj))+np.multiply(y,np.sin(proj))
    y_rot = -np.multiply(x,np.sin(proj))+np.multiply(y,np.cos(proj))
    Isingx = np.sign(x_rot)
    Isingy = np.sign(y_rot)
    S_x = np.absolute(x_rot)
    S_y = np.absolute(y_rot)
    return Isingx, Isingy, S_x, S_y

```

```
# Compose two Ising networks to XY network
def compose(Isingx_new,Isingy_new,proj,S_x, S_y):
    x_rot_new = np.multiply(Isingx_new,S_x)
    y_rot_new = np.multiply(Isingy_new,S_y)
    x_new = np.multiply(x_rot_new,np.cos(proj))-np.multiply(y_rot_new,
    np.sin(proj))
    y_new = np.multiply(x_rot_new,np.sin(proj))+np.multiply(y_rot_new,
    np.cos(proj))
    XY_new = np.arctan2(y_new,x_new)
    return XY_new

def oneMCstepXY(XY):
    proj = np.random.rand()
    [Isingx, Isingy, S_x, S_y] = decompose(XY, proj)
    Isingx_new = oneMCstepIsing(Isingx, S_x)
    Isingy_new = oneMCstepIsing(Isingy, S_y)
    XY_new = compose(Isingx_new, Isingy_new, proj, S_x, S_y)
    return XY_new

# Calculate the energy for XY network
def EnMag(XY):
    energy = 0
    for i in np.arange(L):
        for j in np.arange(L):
            # energy
            energy = energy - (np.cos(XY[i][j]-XY[(i-1)%L][j])+np.cos(
    XY[i][j]-XY[(i+1)%L][j])+np.cos(XY[i][j]-XY[i][(j-1)%L])+np.cos(XY[i
    ][j]-XY[i][(j+1)%L]))
    magx = np.sum(np.cos(XY))
    magy = np.sum(np.sin(XY))
    mag = np.array([magx,magy])
    return energy * 0.5, LA.norm(mag)/(L**2)

# Swendsen Wang method for XY model
def SWang(T):
    XY = init()
    # thermal steps to get the equilibrium
    for step in np.arange(step_r):
        XY = oneMCstepXY(XY)
    # finish with thermal equilibrium, and begin to calc observables
    E_sum = 0
    M_sum = 0
    Esq_sum = 0
    Msq_sum = 0
    lattice_data=np.zeros((step,L,L))
    mag=[]
    for step in np.arange(step):
        XY = oneMCstepXY(XY)
        [E,M] = EnMag(XY)
        mag=np.append(mag,M)
        lattice_data[step]=XY
        E_sum += E
        M_sum += M
        Esq_sum += E**2
        Msq_sum += M**2

    E_mean = E_sum/step/(L**2)
    M_mean = M_sum/step
```

```
    Esq_mean = Esq_sum/step/(L**4)
    Msq_mean = Msq_sum/step

    return lattice_data, XY, E_mean, M_mean, Esq_mean, Msq_mean, mag;

M = np.array([])
E = np.array([])
M_sus = np.array([])
SpcH = np.array([])

Trange = np.linspace(0.75, 1.25, nt)
Lattice_XY=np.zeros((step*nt,L,L))
Mag=[]
for t in range(Trange.shape[0]):
    T=Trange[t]
    lattice_data, Ising, E_mean, M_mean, Esq_mean, Msq_mean, mag =
   SWang(T)
    M = np.append(M, np.abs(M_mean))
    E = np.append(E, E_mean)
    Mag=np.append(Mag,mag, axis=0)
    M_sus = np.append(M_sus, 1/T*(Msq_mean-M_mean**2))
    SpcH = np.append(SpcH, 1/T**2*(Esq_mean-E_mean**2))
    for h in range(step):
        l=t*step+h
        Lattice_XY[l] = lattice_data[h]

    print(T)
    print(lattice_data.shape)
# plot the figures
T = Trange

plt.figure()
plt.plot(T, E, 'rx-')
plt.xlabel(r'Temperature $(\frac{J}{k_B})$')
plt.ylabel(r'$\langle E \rangle$ per site $(J)$')
plt.savefig("E.pdf", format='pdf', bbox_inches='tight')

plt.figure()
plt.plot(T, SpcH, 'kx-')
plt.xlabel(r'Temperature $(\frac{J}{k_B})$')
plt.ylabel(r'$C_V$ per site $(\frac{J^2}{k_B^2})$')
plt.savefig("Cv.pdf", format='pdf', bbox_inches='tight')

plt.figure()
plt.plot(T, M, 'bx-')
plt.xlabel(r'Temperature $(\frac{J}{k_B})$')
plt.ylabel(r'$\langle|M|\rangle$ per site $(\mu)$')
plt.savefig("M.pdf", format='pdf', bbox_inches='tight')

plt.figure()
plt.plot(T, M_sus, 'gx-')
plt.xlabel(r'Temperature $(\frac{J}{k_B})$')
plt.ylabel(r'$\chi$ $(\frac{\mu}{k_B})$')
plt.savefig("chi.pdf", format='pdf', bbox_inches='tight')

plt.tight_layout()
fig = plt.gcf()
plt.show()
```

```python

np.savetxt('output.data',np.c_[T,E,SpcH,M,M_sus])
np.savetxt('mag_data.csv', Mag)
#T = 0.1
#[XY, E_mean, M_mean, Esq_mean, Msq_mean] = SWang(T)
#Cv = 1 / T**2 * (Esq_mean - E_mean**2)
#M_sus = 1 / T * (Msq_mean - M_mean**2)
#[E1,M1] = EnMag(XY)
#E2 = E1/(L**2)
#print(E_mean, E2, M_mean, M1, Cv, M_sus)
## plot the network cluster
#plt.figure()
#plt.matshow(XY,cmap='cool')
#plt.axis('off')
#np.savetxt('latticeXYpredict8_25000.csv',Lattice_XY)

num_lattice=nt*step/n_remove
num_lattice=int(num_lattice)
latticexy=np.zeros((num_lattice,L,L,1))

Lattice_XY=Lattice_XY.reshape((nt*step,L,L,1))
print(Lattice_XY.shape)
k=0
for i in range(Lattice_XY.shape[0]):
  if i%n_remove==0:
    latticexy[k]=Lattice_XY[i]
    k+=1

spin_flatten=np.zeros((Lattice_XY.shape[0],Lattice_XY.shape[1]**2))
for i in range(Lattice_XY.shape[0]):
    spin_flatten[i,:]=Lattice_XY[i,:,:].flatten()

np.savetxt('latticeXY32.csv',spin_flatten)
```

Listing 1.7: dataxy32.py

Convolutional Neural Network

The following code implements the CNN to regenerate the transition temperature for the classical XY model for various lattice sizes.

```python
# -*- coding: utf-8 -*-
"""Copy of cnnXY.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1
   HfqSm4Q3sVTq9yfwMp_tre5QvntrsMVh
"""

# Commented out IPython magic to ensure Python compatibility.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# %matplotlib inline
import keras
from keras.datasets import mnist
```

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.optimizers import Adam
from keras.layers import BatchNormalization
from keras.utils import np_utils
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D,
    GlobalAveragePooling2D
from keras.layers.advanced_activations import LeakyReLU
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import datasets, layers, models

model = models.Sequential()
model.add(layers.Conv2D(12, (3, 3), activation='relu', input_shape
    =(8,8,1)))
#model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(8, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
#model.add(layers.Conv2D(64, (3, 3), activation='relu'))


model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(2))
model.add(Activation('sigmoid'))

model.summary()

model.compile(loss='categorical_crossentropy', optimizer=Adam(),
    metrics=['accuracy'])

from google.colab import drive
drive.mount('/content/drive', force_remount=True)

import pandas as pd
import io
latticexy = pd.read_csv('/content/drive/MyDrive/
    latticeXYpredict16_25000.csv',delimiter=' ', header= None)
latticexy=latticexy.to_numpy()

np.shape(latticexy)

nt=10
step=50000
L=16
x=np.linspace(0.75,1,nt)
temp=np.zeros(step*nt)
for j in range(nt):
  for i in range(step):
    temp[j*step+i]=x[j]

latticexy

temp

'''k=0
for i in range(50000):
  if temp[i]==1:
    k+=1
```

```

print(k)
'''



latticexy=np.reshape(latticexy, (10*step,L,L,1))



print(latticexy.shape)
print(temp.shape)

latticexy_train=latticexy[0:40000]
print(np.shape(latticexy_train))
for i in range(nt):
  if i!=0:
    latticexy_train=np.append(latticexy_train,latticexy[50000*i:50000*i
   +40000],axis=0)
    print(np.shape(latticexy_train))
temp_train=temp[0:40000]
for i in range(10):
  if i!=0:
    temp_train=np.append(temp_train,temp[50000*i:50000*i+40000],axis=0)
    print(temp_train.shape)

temp_train.shape

latticexy_test=latticexy[40000:50000]
for i in range(10):
  if i!=0:
    latticexy_test=np.append(latticexy_test,latticexy[50000*i
   +40000:50000*i+50000],axis=0)
temp_test=temp[40000:50000]
for i in range(10):
  if i!=0:
    temp_test=np.append(temp_test,temp[50000*i+40000:50000*i+50000],
   axis=0)

temp_train[temp_train<0.892]=0

temp_train[temp_train>0.892]=1

temp_test[temp_test>0.892]=1
temp_test[temp_test<0.892]=0

plt.imshow(latticexy_test[1990,:,:,0])
plt.show

temp_train.shape

temperature_train=np.zeros((temp_train.shape[0],2))
temperature_test=np.zeros((temp_test.shape[0],2))
for i in range(temp_train.shape[0]):
  if temp_train[i]==0:
    temperature_train[i]=[1,0]
  else:
    temperature_train[i]=[0,1]
```

```python

for i in range(temp_test.shape[0]):
  if temp_test[i]==0:
    temperature_test[i]=[1,0]
  else:
    temperature_test[i]=[0,1]

latticexy_test/=2*np.pi
latticexy_train/=2*np.pi

#model.fit(latticexy_train, temp_train, epochs=5, validation_data=tuple
    (latticexy_test,temp_test))
model.fit(latticexy_train, temperature_train, batch_size=64, epochs=50,
    validation_data=(latticexy_test,temperature_test))

score = model.evaluate(latticexy_test,temperature_test)
print()
print('Test accuracy: ', score[1])

predictions = model.predict(latticexy_test)

t = np.linspace(0.75, 1, 10)

y=np.zeros((10,1000,2))
for i in range(10):
  y[i]=predictions[1000*i:1000*(i+1)]

y_avg=np.mean(y, axis=1)
y_avg_sum=np.mean(y_avg, axis=0)

y_avg.shape

line=np.full(10,0.8935)
y_line=np.linspace(0, 1,10)

plt.plot(t,y_avg[:,0])
plt.plot(t,y_avg[:,1])
plt.plot(line,y_line)
```

Listing 1.8: cnnxy.py

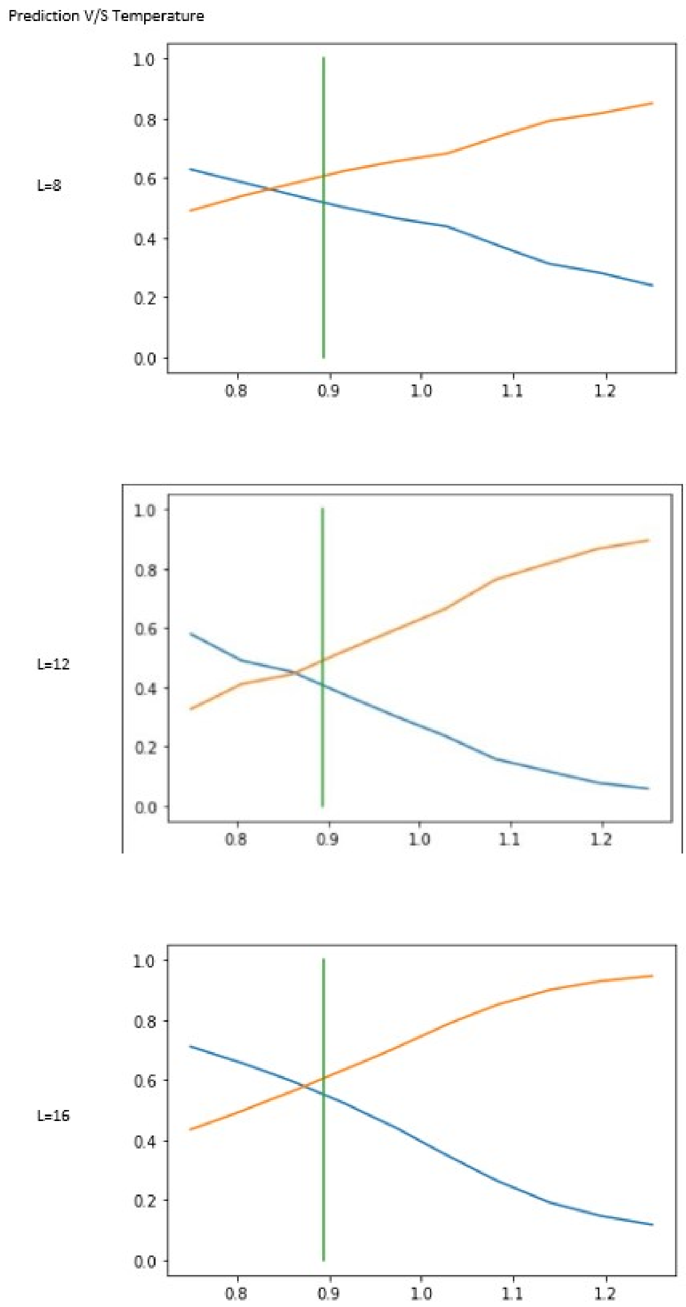

Figure 1.8: Transition temperature with CNN