Classical Spin System and Kosterlitz Thouless Phase Transition

Shang Zhang

Introduction

In this project, I will study the system of classical 2-D XY model with cluster Monte Carlo method.
In 2-D XY model, there is a phase transition from bound vortex-antivortex pairs at low temperatures to unpaired vortices and anti-vortices at some critical temperature, with the name "Kosterlitz–Thouless transition (KT transition)" [1]. Work on the transition led to the 2016 Nobel Prize in Physics being awarded to Thouless, Kosterlitz and Duncan Haldane.
The Hamiltonian for 2-D XY model is given from the nearest neighbor interaction inside lattice:

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} \cos\left(\theta_i - \theta_j\right)$$

J describes the coupling between spins, and is given $J = 1$ in my simulation, which illustrates the ferromagnetic condition. θ_i is the angle of spin i, with nearest neighbor $\langle i, j \rangle$ interaction only considered. My simulation lattice is a square lattice with periodic boundary condition.

Methodology

Cluster Algorithm

Here I will use $O(N)$ method, which is a kind of cluster Monte Carlo method to simulate the 2-D XY spin system. $O(N)$ algorithm is one kind of cluster algorithm based on the Swendsen-Wang algorithm, which comes from the basic idea for spin flipping in: Instead of flipping single spins we propose to flip big clusters of spins and choose them in a clever way so that the probability of flipping these clusters is large.
$O(N)$ algorithm is given as:

while *Inside the Loop for Monte Carlo Updating* **do**
  Project all spins onto a random direction $\hat{e}$;
  while *Go through all the nearest neighbored spins* $\langle S_i, S_j \rangle$ *inside lattice* **do**
    if $\hat{e}S_i$ *and* $\hat{e}S_j$ *are aligned* **then**
      Connect $(\hat{e}S_i)(\hat{e}S_j)$ with probability $1 - \exp\left(-2\beta J(\hat{e}S_i)(\hat{e}S_j)\right)$
    else
      Never connect $(\hat{e}S_i)(\hat{e}S_j)$;
    end
  end
  Do cluster labeling (Use Hoshen-Kopelman algorithm);
  Measurements are performed;
  Flip cluster spins by inverting the projection on the $\hat{e}$ direction with probability 1/2
end

Algorithm 1: $O(N)$ Algorithm

In the 2-D XY model, $N = 2$, so the random direction $\hat{e}$ is randomly chosen in 2-D plane ($\theta_{\hat{e}} \in [-\pi, \pi]$).

Observables

The physics quantities (per site) to measure by my cluster Monte Carlo simulation, (after having reached the thermal equilibrium)

$$M = \frac{1}{N}\sum_{i=1}^{N} S_i$$
$$E = -\frac{1}{N}J\sum_{\langle i,j\rangle} S_i \cdot S_j$$
$$\chi = \frac{1}{k_B T}[\langle M^2\rangle - \langle |M|\rangle^2]$$
$$C_V = \frac{1}{k_B^2 T^2}[\langle E^2\rangle - \langle E\rangle^2]$$

Vortex

In the 2-D XY model, vortices are topologically stable configurations. Vortex generation becomes thermodynamically favorable at the critical temperature T_c of the KT transition. At temperatures below this, vortex generation has a power law correlation. The figures attached are examples for vortices inside 2-D XY model, from website [2].

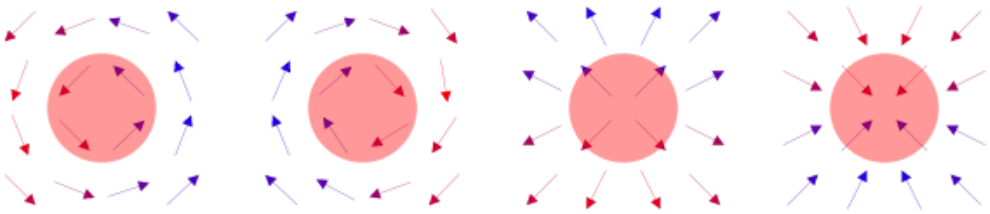

Figure 1: Vortices, from left to right: center left, center right, source, sink

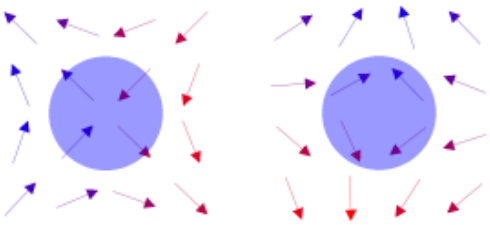

Figure 2: Anti-vortices, both are saddles

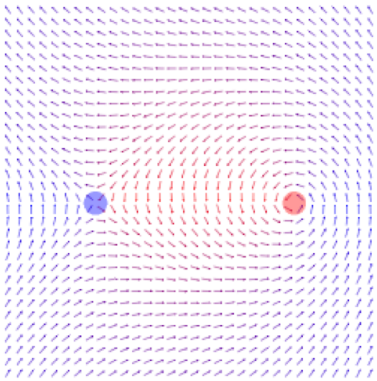

Figure 3: a vortex pair

Simulation Results

Observables

My simulation is given for $J > 0$, which is the ferromagnetic case. The system size is from 8×8 to 64×64, with 2000 steps to reach the thermal equilibrium. Then the calculation for the physical quantities is given with 20000 steps after reaching the thermal equilibrium. So we can get the results as,

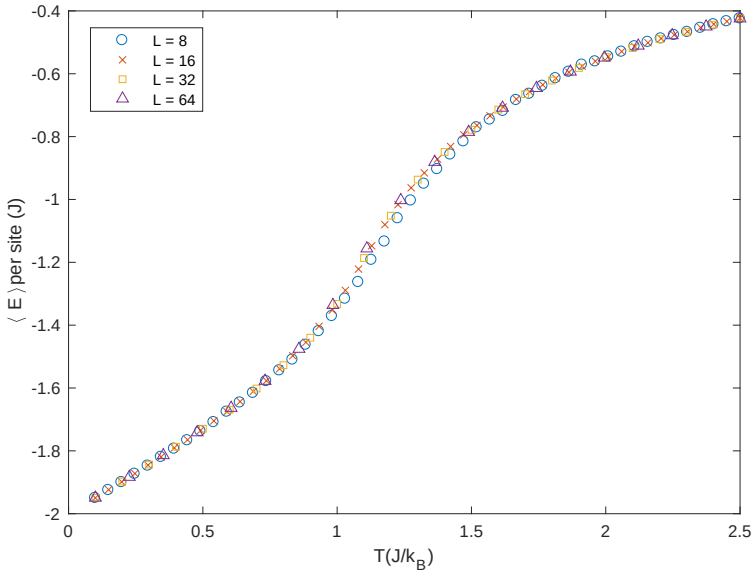

Figure 4: Energy Per Site

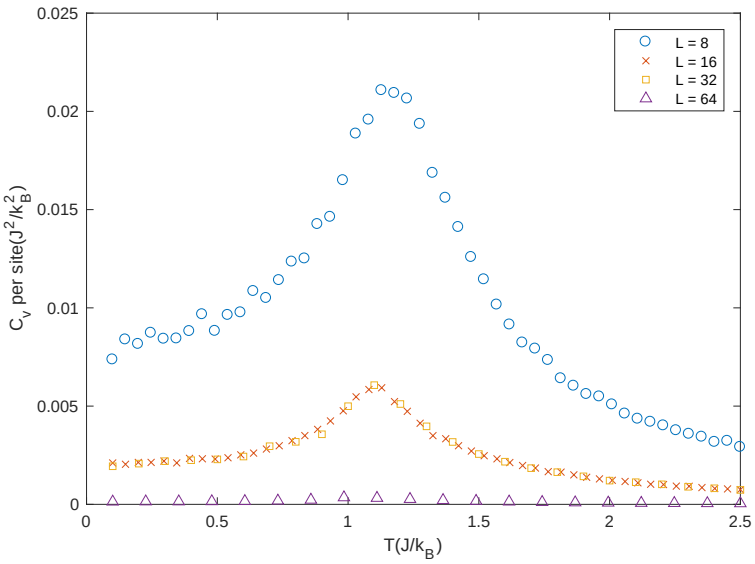

Figure 5: Specific Heat Per Site

The peak for $L = 64$ is not very clear above, so I pick this out as,

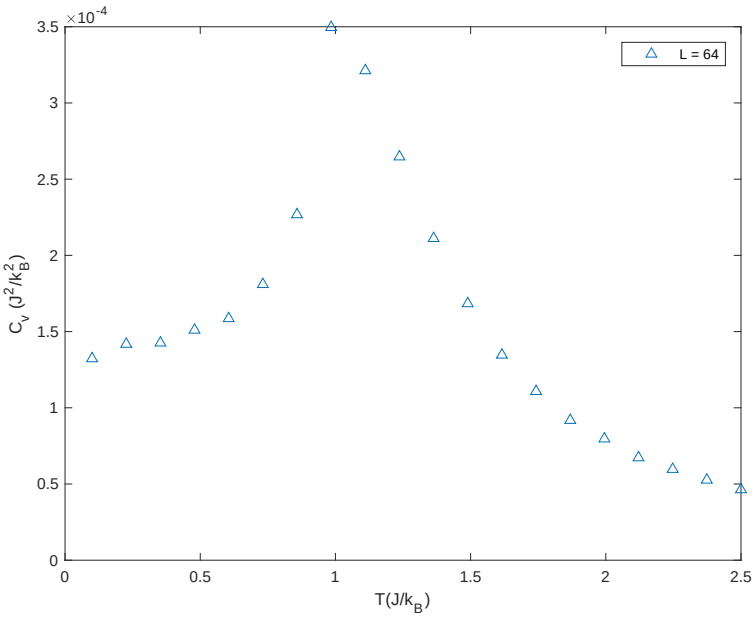

Figure 6: Specific Heat Per Site for $L = 64$

Then the other observables,

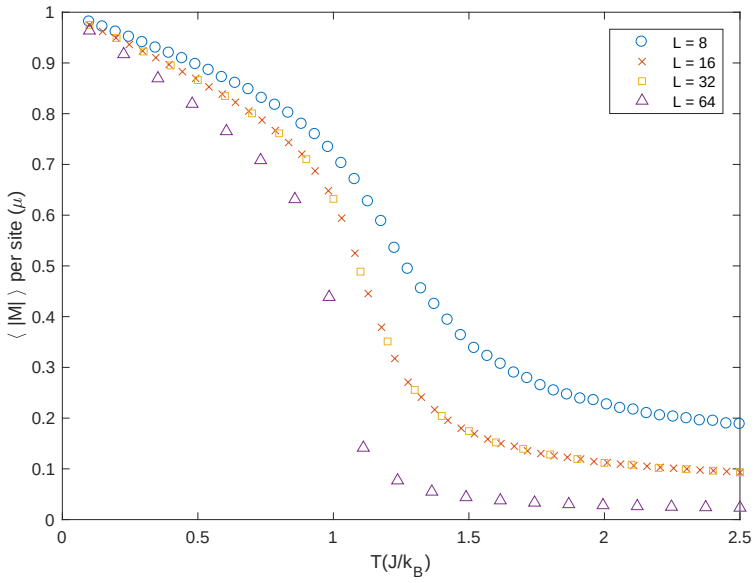

Figure 7: Abusolute Value of Magnetization Per Site

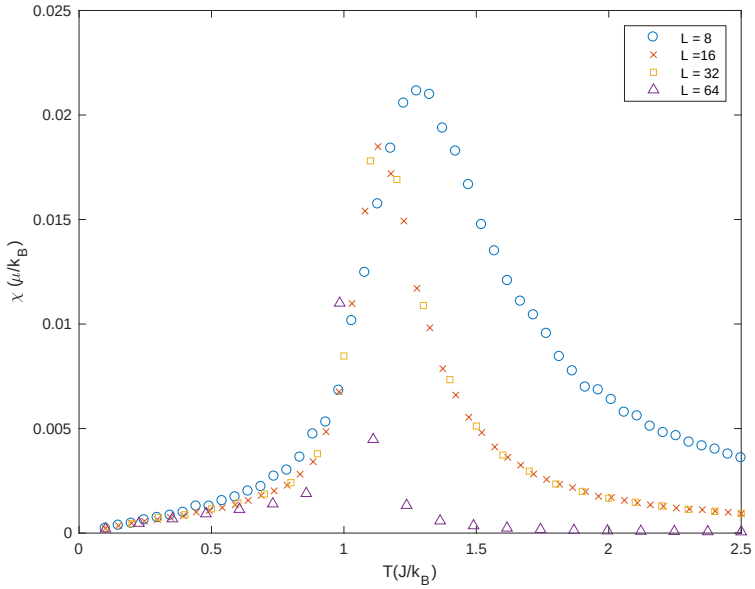

Figure 8: Magnetic Susceptibility Per Site

Clearly we can see KT phase transition since there is peak in C_v and χ for XY spin system. Look at the finite size effect for the critical temperature T_c:

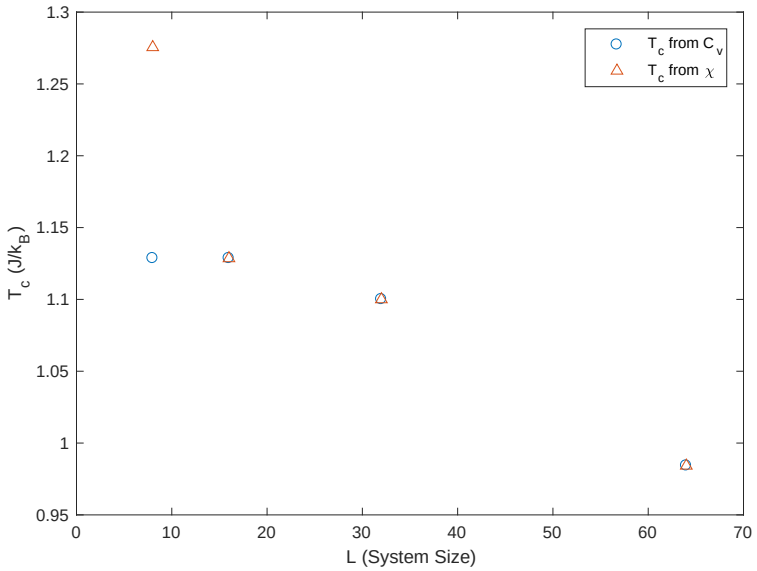

Figure 9: Finite Size Effect for T_c

So we can find the location of the phase transition approaches the limit for a infinitely large lattice: $T_c = 0.89213(10)(\frac{J}{k_B})$ [3].

Vortex

Now look at the spin configurations after reaching the thermal equilibrium for the XY lattice system (System Size $L = 16$) and pick out the vortices (red) and anti-vortices (blue). The thermal equilibrium is got from 3000 steps of Monte Carlo updates.

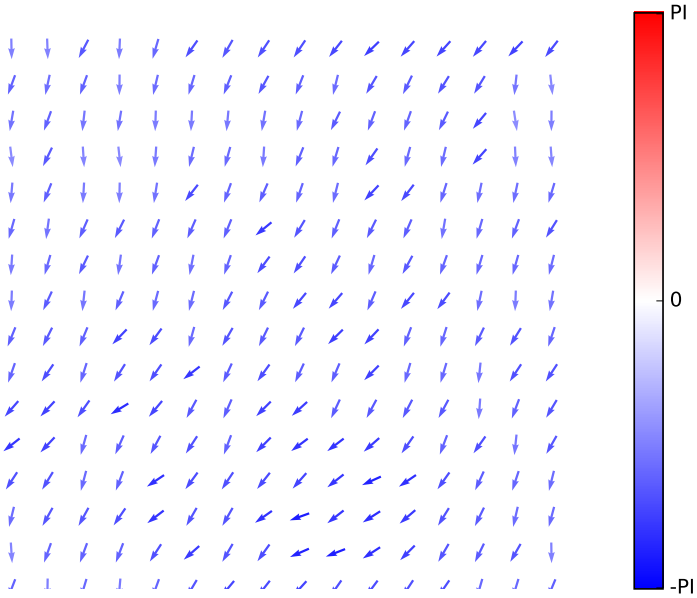

Figure 10: Spin Configuration for $T = 0.10(\frac{J}{k_B})$

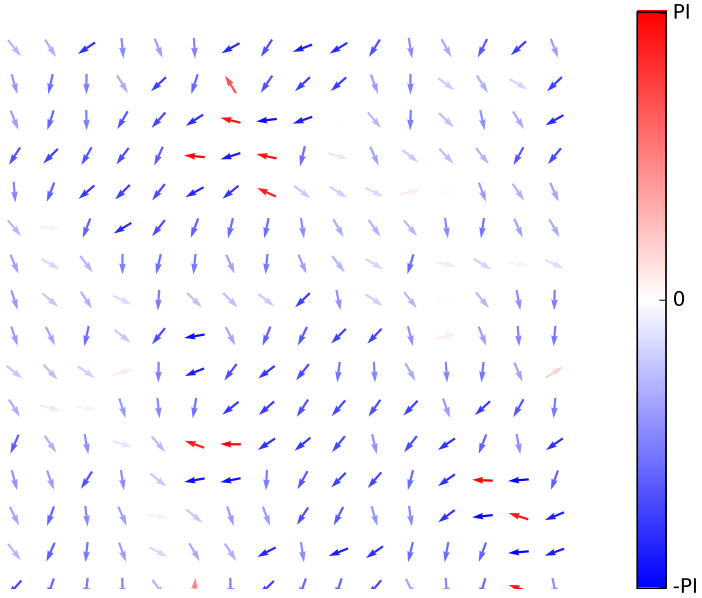

Figure 11: Spin Configuration for $T = 0.98(\frac{J}{k_B})$

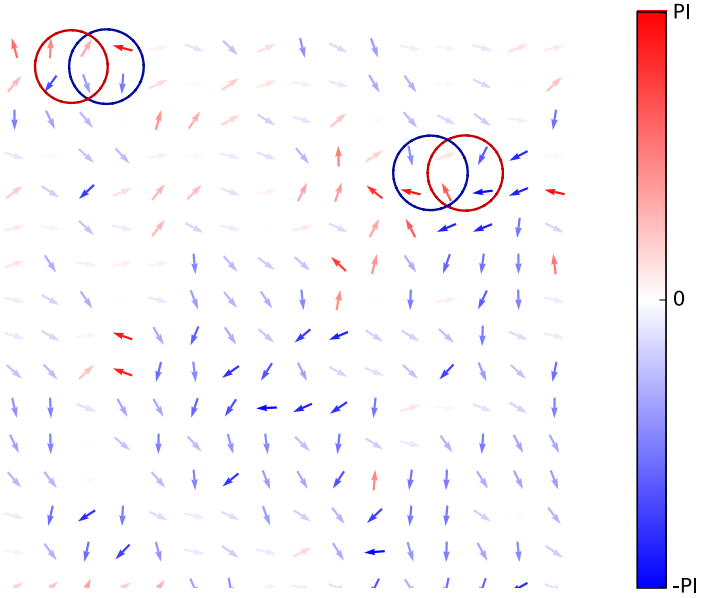

Figure 12: Spin Configuration for $T = 1.11(\frac{J}{k_B})$

So we can see that the vortex pairs are beginning to detach near critical temperature $T_c = 1.129(\frac{J}{k_B})$ (For $L = 16$).

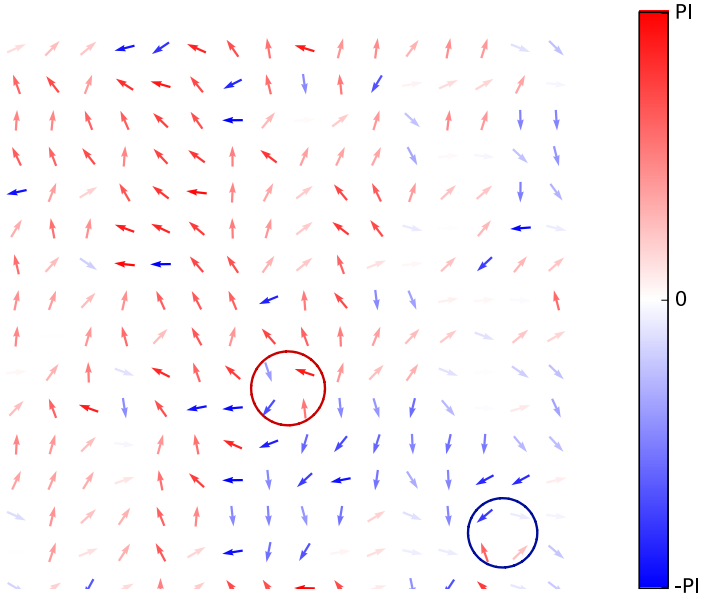

Figure 13: Spin Configuration for $T = 1.24(\frac{J}{k_B})$

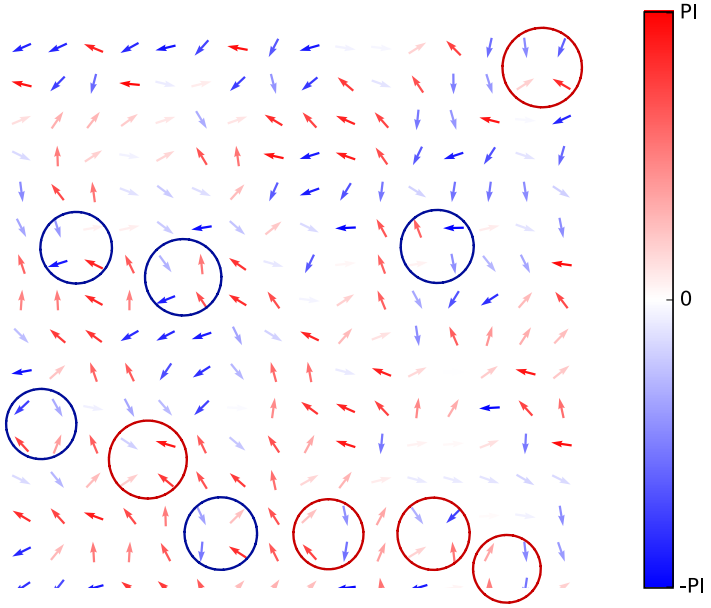

Figure 14: Spin Configuration for $T = 2.50(\frac{J}{k_B})$

From the peak in C_v, the critical temperature of this system size is $T_c = 1.129(\frac{J}{k_B})$. Same temperature is gotten from χ in my simulation.
So when $T < T_c$, there are no vortices. When $T > T_c$, vortices begin to appear and usually they will appear in pairs. With the increase of the system temperature, the amount of vortices pairs is also increasing.

Discussion

Mermin–Wagner Theorem

The Mermin–Wagner theorem states that continuous symmetries cannot be spontaneously broken at finite temperature in systems with sufficiently short-range interactions in dimensions $d \leq 2$ [4].
The 2-D XY model's KT phase transition is a special example for Mermin–Wagner theorem.
The Mermin–Wagner theorem prevents the 2-D XY model system to have any spontaneously symmetry broken. However, the KT phase transition is not associated with spontaneously symmetry broken. As a result, **Phase Transition $\neq$ Symmetry Breaking**.
Then Kosterlitz and Thouless proposed that topological defects could explain the phase transition. The 2-D XY system is not expected to have a second-order phase transition. However, one finds a low-temperature quasi-ordered phase with a correlation function that decreases with the distance like a power, which depends on the temperature. The transition from the high-temperature disordered phase with the exponential correlation to this low-temperature quasi-ordered phase is a Kosterlitz–Thouless transition. It is a phase transition of infinite order.

References

[1] John Michael Kosterlitz and David James Thouless. Ordering, metastability and phase transitions in two-dimensional systems. *Journal of Physics C: Solid State Physics*, 6(7):1181, 1973.

[2] Tobias Rautenkranz Alexander Käser, Tobias Maier. The 2d xy model, 2007.

[3] Peter Olsson. Monte carlo analysis of the two-dimensional xy model. ii. comparison with the kosterlitz renormalization-group equations. *Physical Review B*, 52(6):4526, 1995.

[4] N David Mermin and Herbert Wagner. Absence of ferromagnetism or antiferromagnetism in one-or two-dimensional isotropic heisenberg models. *Physical Review Letters*, 17(22):1133, 1966.

Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: zhshang
"""
import matplotlib
matplotlib.use('Agg')
import numpy as np
from numpy import linalg as LA
import matplotlib.pyplot as plt

L = 16
ESTEP = 1000
STEP = 10000

J = 1 # J>0 to make it ferromagnetic

# Intitialize the XY network
def Init():
```

```
20      return np.random.rand(L, L)*2*np.pi
21      #return np.ones([L, L])
22
23  # periodic neighbor
24  def next(x):
25      if x == L-1:
26          return 0
27      else:
28          return x+1
29
30  # construct the bond lattice
31  def FreezeBonds(Ising,T,S):
32      iBondFrozen = np.zeros([L,L])
33      jBondFrozen = np.zeros([L,L])
34      for i in np.arange(L):
35          for j in np.arange(L):
36              freezProb_nexti = 1 - np.exp(-2 * J * S[i][j] * S[next(i)][j] / T)
37              freezProb_nextj = 1 - np.exp(-2 * J * S[i][j] * S[i][next(j)] / T)
38              if (Ising[i][j] == Ising[next(i)][j]) and (np.random.rand() <
                    freezProb_nexti):
39                  iBondFrozen[i][j] = 1
40              if (Ising[i][j] == Ising[i][next(j)]) and (np.random.rand() <
                    freezProb_nextj):
41                  jBondFrozen[i][j] = 1
42      return iBondFrozen, jBondFrozen
43
44  # H-K algorithm to identify clusters
45  def properlabel(prp_label,i):
46      while prp_label[i] != i:
47          i = prp_label[i]
48      return i
49
50  # Swendsen-Wang cluster
51  def clusterfind(iBondFrozen,jBondFrozen):
52      cluster = np.zeros([L, L])
53      prp_label = np.zeros(L**2)
54      label = 0
55      for i in np.arange(L):
56          for j in np.arange(L):
57              bonds = 0
58              ibonds = np.zeros(4)
59              jbonds = np.zeros(4)
60
61              # check to (i-1,j)
62              if (i > 0) and iBondFrozen[i-1][j]:
63                  ibonds[bonds] = i-1
64                  jbonds[bonds] = j
65                  bonds += 1
66              # (i,j) at i edge, check to (i+1,j)
67              if (i == L-1) and iBondFrozen[i][j]:
68                  ibonds[bonds] = 0
69                  jbonds[bonds] = j
70                  bonds += 1
```

```
71              # check to (i,j-1)
72              if (j > 0) and jBondFrozen[i][j-1]:
73                  ibonds[bonds] = i
74                  jbonds[bonds] = j-1
75                  bonds += 1
76              # (i,j) at j edge, check to (i,j+1)
77              if (j == L-1) and jBondFrozen[i][j]:
78                  ibonds[bonds] = i
79                  jbonds[bonds] = 0
80                  bonds += 1
81
82              # check and label clusters
83              if bonds == 0:
84                  cluster[i][j] = label
85                  prp_label[label] = label
86                  label += 1
87              else:
88                  minlabel = label
89                  for b in np.arange(bonds):
90                      plabel = properlabel(prp_label,cluster[ibonds[b]][jbonds[b]])
91                      if minlabel > plabel:
92                          minlabel = plabel
93
94                  cluster[i][j] = minlabel
95                  # link to the previous labels
96                  for b in np.arange(bonds):
97                      plabel_n = cluster[ibonds[b]][jbonds[b]]
98                      prp_label[plabel_n] = minlabel
99                      # re-set the labels on connected sites
100                     cluster[ibonds[b]][jbonds[b]] = minlabel
101     return cluster, prp_label
102
103 # flip the cluster spins
104 def flipCluster(Ising,cluster,prp_label):
105     for i in np.arange(L):
106         for j in np.arange(L):
107             # relabel all the cluster labels with the right ones
108             cluster[i][j] = properlabel(prp_label,cluster[i][j])
109     sNewChosen = np.zeros(L**2)
110     sNew = np.zeros(L**2)
111     flips = 0 # get the number of flipped spins to calculate the Endiff and Magdiff
112     for i in np.arange(L):
113         for j in np.arange(L):
114             label = cluster[i][j]
115             randn = np.random.rand()
116             # mark the flipped label, use this label to flip all the cluster elements
                    with this label
117             if (not sNewChosen[label]) and randn < 0.5:
118                 sNew[label] = +1
119                 sNewChosen[label] = True
120             elif (not sNewChosen[label]) and randn >= 0.5:
121                 sNew[label] = -1
122                 sNewChosen[label] = True
```

```
123
124             if Ising[i][j] != sNew[label]:
125                 Ising[i][j] = sNew[label]
126                 flips += 1
127
128     return Ising,flips
129
130 # Swendsen-Wang Algorithm in Ising model (with coupling constant dependency on sites)
131 # One-step for Ising
132 def oneMCstepIsing(Ising, S):
133     [iBondFrozen, jBondFrozen] = FreezeBonds(Ising, T, S)
134     [SWcluster, prp_label] = clusterfind(iBondFrozen, jBondFrozen)
135     [Ising, flips] = flipCluster(Ising, SWcluster, prp_label)
136     return Ising
137
138 # Decompose XY network to two Ising networks with project direction proj
139 def decompose(XY,proj):
140     x = np.cos(XY)
141     y = np.sin(XY)
142     x_rot = np.multiply(x,np.cos(proj))+np.multiply(y,np.sin(proj))
143     y_rot = -np.multiply(x,np.sin(proj))+np.multiply(y,np.cos(proj))
144     Isingx = np.sign(x_rot)
145     Isingy = np.sign(y_rot)
146     S_x = np.absolute(x_rot)
147     S_y = np.absolute(y_rot)
148     return Isingx, Isingy, S_x, S_y
149
150 # Compose two Ising networks to XY network
151 def compose(Isingx_new,Isingy_new,proj,S_x, S_y):
152     x_rot_new = np.multiply(Isingx_new,S_x)
153     y_rot_new = np.multiply(Isingy_new,S_y)
154     x_new = np.multiply(x_rot_new,np.cos(proj))-np.multiply(y_rot_new,np.sin(proj))
155     y_new = np.multiply(x_rot_new,np.sin(proj))+np.multiply(y_rot_new,np.cos(proj))
156     XY_new = np.arctan2(y_new,x_new)
157     return XY_new
158
159 def oneMCstepXY(XY):
160     proj = np.random.rand()
161     [Isingx, Isingy, S_x, S_y] = decompose(XY, proj)
162     Isingx_new = oneMCstepIsing(Isingx, S_x)
163     #Isingy_new = oneMCstepIsing(Isingy, S_y)
164     # Here we only use the flopping of Ising in projection direction, without the
        perpendicular direction
165     #XY_new = compose(Isingx_new, Isingy_new, proj, S_x, S_y)
166     XY_new = compose(Isingx_new, Isingy, proj, S_x, S_y)
167     return XY_new
168
169 # Calculate the energy for XY network
170 def EnMag(XY):
171     energy = 0
172     for i in np.arange(L):
173         for j in np.arange(L):
174             # energy
```

```
175             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]))
176     magx = np.sum(np.cos(XY))
177     magy = np.sum(np.sin(XY))
178     mag = np.array([magx,magy])
179     return energy * 0.5, LA.norm(mag)/(L**2)
180
181 # Swendsen Wang method for XY model
182 def SWang(T):
183     XY = Init()
184     # thermal steps to get the equilibrium
185     for step in np.arange(ESTEP):
186         XY = oneMCstepXY(XY)
187     # finish with thermal equilibrium, and begin to calc observables
188     E_sum = 0
189     M_sum = 0
190     Esq_sum = 0
191     Msq_sum = 0
192     for step in np.arange(STEP):
193         XY = oneMCstepXY(XY)
194         [E,M] = EnMag(XY)
195
196         E_sum += E
197         M_sum += M
198         Esq_sum += E**2
199         Msq_sum += M**2
200
201     E_mean = E_sum/STEP/(L**2)
202     M_mean = M_sum/STEP
203     Esq_mean = Esq_sum/STEP/(L**4)
204     Msq_mean = Msq_sum/STEP
205
206     return XY, E_mean, M_mean, Esq_mean, Msq_mean
207
208 M = np.array([])
209 E = np.array([])
210 M_sus = np.array([])
211 SpcH = np.array([])
212 Trange = np.linspace(0.1, 2.5, 10)
213 for T in Trange:
214     [Ising, E_mean, M_mean, Esq_mean, Msq_mean] = SWang(T)
215     M = np.append(M, np.abs(M_mean))
216     E = np.append(E, E_mean)
217     M_sus = np.append(M_sus, 1/T*(Msq_mean-M_mean**2))
218     SpcH = np.append(SpcH, 1/T**2*(Esq_mean-E_mean**2))
219
220 # plot the figures
221 T = Trange
222
223 plt.figure()
224 plt.plot(T, E, 'rx-')
225 plt.xlabel(r'Temperature $(\frac{J}{k_B})$')
```

```
226 plt.ylabel(r'$\langle E \rangle$ per site $(J)$')
227 plt.savefig("E.pdf", format='pdf', bbox_inches='tight')
228
229 plt.figure()
230 plt.plot(T, SpcH, 'kx-')
231 plt.xlabel(r'Temperature $(\frac{J}{k_B})$')
232 plt.ylabel(r'$C_V$ per site $(\frac{J^2}{k_B^2})$')
233 plt.savefig("Cv.pdf", format='pdf', bbox_inches='tight')
234
235 plt.figure()
236 plt.plot(T, M, 'bx-')
237 plt.xlabel(r'Temperature $(\frac{J}{k_B})$')
238 plt.ylabel(r'$\langle|M|\rangle$ per site $(\mu)$')
239 plt.savefig("M.pdf", format='pdf', bbox_inches='tight')
240
241 plt.figure()
242 plt.plot(T, M_sus, 'gx-')
243 plt.xlabel(r'Temperature $(\frac{J}{k_B})$')
244 plt.ylabel(r'$\chi$ $(\frac{\mu}{k_B})$')
245 plt.savefig("chi.pdf", format='pdf', bbox_inches='tight')
246
247 plt.tight_layout()
248 fig = plt.gcf()
249 plt.show()
250
251 np.savetxt('output.data',np.c_[T,E,SpcH,M,M_sus])
252
253 # for T in np.linspace(0.1, 2.5, 20):
254 #     [XY, E_mean, M_mean, Esq_mean, Msq_mean] = SWang(T)
255 #     # plot the network cluster
256 #     [X, Y] = np.mgrid[0:L, 0:L]
257 #     X_plot = np.cos(XY)
258 #     Y_plot = np.sin(XY)
259
260 #     plt.figure()
261 #     plt.quiver(X,Y,X_plot,Y_plot,XY,pivot='mid',cmap=plt.cm.bwr, clim=[-3.15,3.15])
262 #     plt.axis('equal')
263 #     plt.axis('off')
264 #     cbar = plt.colorbar(ticks=[-3.14, 0, 3.14])
265 #     cbar.ax.set_yticklabels(['-PI', '0', 'PI'])
266
267 #     name = 'XY-T'+str(int(T*1000)).zfill(4)
268 #     figname = name+'.pdf'
269 #     plt.savefig(figname, format='pdf', bbox_inches='tight')
270
271 #     filename = name+'.data'
272 #     np.savetxt(filename,XY
```