import numpy as np

from numpy import \*

import matplotlib.pyplot as plt

import random

# get indices of neighbors of node at index i,j

def get\_neigh(i, j):

# look at neighbors up, down, left, right

neigh = [None]\*4

neigh[0] = [i-1,j]

neigh[1] = [i+1,j]

neigh[2] = [i,j-1]

neigh[3] = [i,j+1]

# deal with edge cases

if i == 0: # upper left of graph

neigh[0] = [6,j]

elif i == 6: # upper right of graph

neigh[1] = [0,j]

if j == 0: # lower left of graph

neigh[2] = [i,6]

elif j == 6: # lower right of graph

neigh[3] = [i,0]

return neigh

def make\_donut\_graph(num\_nodes):

# make a 49x49 sized matrix

# for each node n in {1,...49}, store 1 for each corresponding neighbor

graph = np.zeros((num\_nodes, num\_nodes))

# index row and column into 2D grid

for i in range(7):

for j in range(7):

# get number of node from location in 3D grid

node\_idx = i\*7+j

neigh = get\_neigh(i,j)

# for each node in graph, set it's neighbor's values to 1

for n in range(len(neigh)):

ni = neigh[n][0]

nj = neigh[n][1]

neigh\_idx = ni\*7+nj

graph[node\_idx, neigh\_idx] = 1

return graph

def gibbs(graph, num\_nodes, theta\_st, theta\_s, burn\_in, num\_samps):

# store for each of the {1,...49} nodes, it's corresponding samples

samples = np.zeros((num\_nodes, num\_samps))

# initialize X\_est values to random -1, 0 or 1 value

X\_est = np.ones((num\_nodes,1))

for i in range(num\_nodes):

X\_est[i] = np.random.randint(-1,1)

for i in range(burn\_in + num\_samps):

for n in range(num\_nodes):

# get neighbors of node

n\_neigh = X\_est[graph[n] != 0]

p = theta\_s[n] + theta\_st \* np.sum(n\_neigh)

rand\_num = np.random.rand()

if rand\_num < np.exp(p)/(np.exp(p)+np.exp(-p)):

X\_est[n] = 1

else:

X\_est[n] = -1

# record samples after burn-in period

if(i > burn\_in):

for j in range(num\_nodes):

samples[j, i-burn\_in] = X\_est[j]

return samples

def gibbs\_mean(gibbs\_samps, num\_samps, num\_nodes):

mean = np.zeros((num\_nodes, 1))

for i in range(num\_nodes):

mean[i] = sum(gibbs\_samps[i,:])/num\_samps

mean\_formatted = np.zeros((7,7))

n = 0

for i in range(7):

for j in range(7):

mean\_formatted[i][j] = mean[n]

n += 1

return mean\_formatted

def naive\_mean(graph, num\_nodes, theta\_st, theta\_s):

X\_est = np.random.rand(num\_nodes,1)

thresh = 0.0000000001

curr\_diff = 10

# keep iterating until converged

while(curr\_diff > thresh):

for n in range(num\_nodes):

prev\_X\_est = X\_est

n\_neigh = X\_est[graph[n] != 0]

p = theta\_s[n] + theta\_st \* np.sum(n\_neigh)

X\_est[n] = (np.exp(2\*p) - 1)/(np.exp(2\*p) + 1)

diff = X\_est - prev\_X\_est

curr\_diff = np.sum(diff)/num\_nodes

# reshape to be 7x7 matrix

X\_est\_formatted = np.zeros((7,7))

n = 0

for i in range(7):

for j in range(7):

X\_est\_formatted[i][j] = X\_est[n]

n += 1

return X\_est\_formatted

def compute\_l1\_dist(gibbs\_mean, naive\_samps):

dist = 0

for i in range(7):

for j in range(7):

dist += abs(naive\_samps[i][j] - gibbs\_mean[i][j])

dist /= 49.0

return dist

if \_\_name\_\_ == "\_\_main\_\_":

num\_nodes = 49

# edge and node compatibility functions

theta\_st = 0.25

theta\_s = [(-1.0)\*\*s for s in range(1,num\_nodes+1)]

# make donut graph

graph = make\_donut\_graph(num\_nodes)

burn\_in = 1000

num\_samps = 5000

# run gibbs sampler

print "(a) Gibbs Sampler Results"

gibbs\_samps = gibbs(graph, num\_nodes, theta\_st, theta\_s, burn\_in, num\_samps)

gibbs\_mean = gibbs\_mean(gibbs\_samps, num\_samps, num\_nodes)

print gibbs\_mean

# run naive mean field sampler

print "(b) Naive Mean Field Results"

naive\_samps = naive\_mean(graph, num\_nodes, theta\_st, theta\_s)

print naive\_samps

dist = compute\_l1\_dist(gibbs\_mean, naive\_samps)

print "Average l1 dist btwn mean field and gibbs estimates: ", dist

(a) Gibbs Sampler Results

[[-0.7558 0.5362 -0.5774 0.5802 -0.5778 0.5434 -0.7598]

[ 0.547 -0.4058 0.4234 -0.4014 0.4166 -0.3826 0.547 ]

[-0.571 0.4154 -0.4578 0.4326 -0.4266 0.4414 -0.5626]

[ 0.583 -0.429 0.3918 -0.4506 0.4198 -0.4354 0.5794]

[-0.581 0.4262 -0.4206 0.4358 -0.467 0.3922 -0.5794]

[ 0.5346 -0.405 0.4186 -0.421 0.4098 -0.4174 0.5234]

[-0.7702 0.5198 -0.5842 0.5618 -0.5846 0.5358 -0.7578]]

(b) Naive Mean Field Results

[[-0.41346074 0.86844991 -0.59719016 0.8713221 -0.40822075 0.85263028 -0.5776518 ]

[ 0.78678465 -0.40633878 0.75207555 -0.27629648 0.79235679 -0.48092917 0.76274713]

[-0.53229605 0.75846609 -0.4093037 0.77970742 -0.36170232 0.73793536 -0.55413692]

[ 0.87670109 -0.44620729 0.79824753 -0.2370104 0.75652047 -0.35467228 0.81269737] [-0.4609323 0.74304278 -0.21930455 0.74754156 -0.40820033 0.7809346 -0.49383011] [ 0.82332466 -0.28673541 0.72194475 -0.34330754 0.76423229 -0.40640367 0.83615792] [-0.50190584 0.77332098 -0.51476292 0.8012062 -0.60689747 0.82974688 -0.69284841]]

Average l1 dist btwn mean field and gibbs estimates: 0.202458666649