Stat 202C - Project 3

Jiayu Wu | 905054229

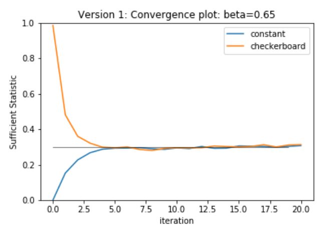
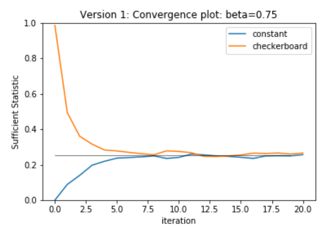
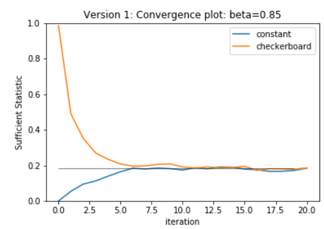
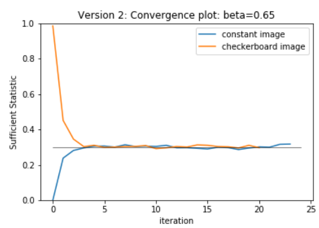
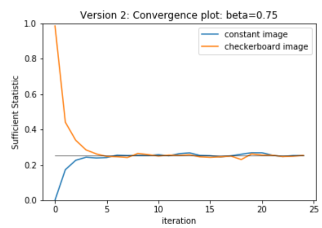
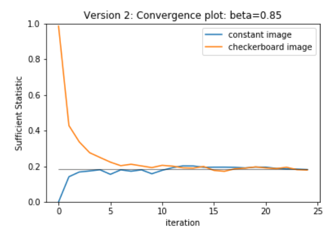
Sufficient statistics

To alleviate the effect of randomness, we run gibbs sampler as in Project2 for 10 times and compute the average:

β	0.65	0.75	0.85
mean	0.29717	0.25105	0.18252

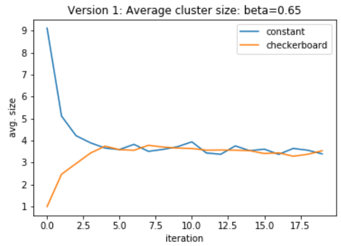
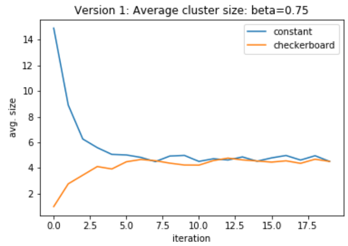
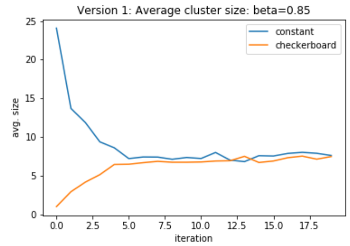
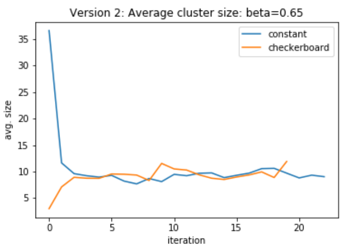
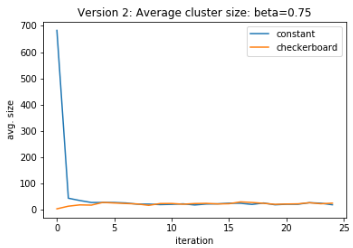
Iterations are count in sweep in the following results

1, 2.

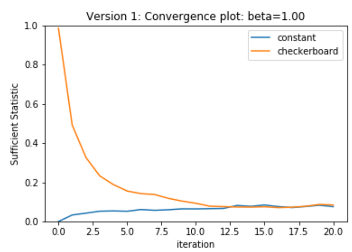
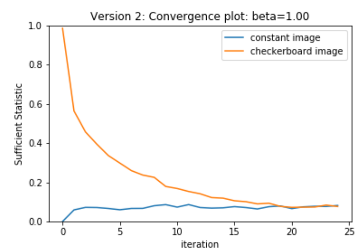
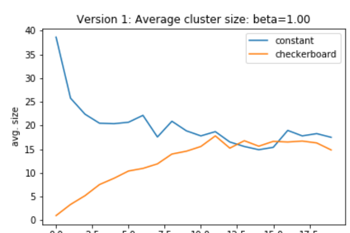
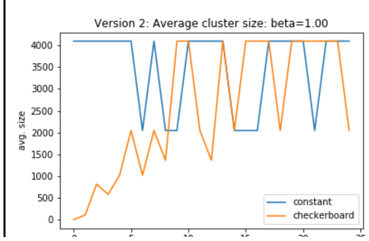
β	0.65	0.75	0.85
Version 1	-	-	-
Convergence Plot			
MC1 Convergence Time ($\epsilon = 0.005$)	5	8	6
MC2 Convergence Time ($\epsilon = 0.005$)	4	8	11
h^*	0.297	0.251	0.183
Coalescence Time	5	13	11
Coalescence $H(x)$	0.293	0.249	0.187
-	-	-	-
Version 2	-	-	-
Convergence Plot			
MC1 Convergence Time ($\epsilon = 0.005$)	3	6	4
MC2 Convergence Time ($\epsilon = 0.005$)	5	5	17
Coalescence Time	6	11	14
Coalescence $H(x)$	0.300	0.251	0.194

3.

β	0.65	0.75	0.85
Version 1	-	-	-

Avg. Size Plot			
-	-	-	-
Version 2	-	-	-
Avg. Size Plot			

4.

$\beta = 1.0$	**Version 1**	**Version 2**
Convergence Plot		
Convergence Time	14	19
Convergence $H(X)$	0.077	0.079
Avg. Size Plot		

In [ ]:

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

In [ ]:
```python
def margin(x, y, mat, beta):
    one = 0
    zero = 0
    if x > 0:
        one += mat[x-1, y]
        zero += 1 - mat[x-1, y]
    if x < n-1:
        one += mat[x+1, y]
        zero += 1 - mat[x+1, y]
    if y > 0:
        one += mat[x, y-1]
        zero += 1 - mat[x, y-1]
    if y < n-1:
        one += mat[x, y+1]
        zero += 1 - mat[x, y+1]
    c = np.exp(beta*one)
    prob = c/(c+np.exp(beta*zero))
    return prob
def swp(white, black, beta):
    for x in range(n):
        for y in range(n):
            rdn = np.random.uniform(0,1)
            if rdn < margin(x, y, white, beta):
                white[x, y] = 1
            else:
                white[x, y] = 0
            if rdn < margin(x, y, black, beta):
                black[x, y] = 1
            else:
                black[x, y] = 0
def suff_stat(img):
    n = img.shape[0]
    S = sum(sum(img[1:,:]-img[:(n-1),:]!=0))+sum(sum(img[:,1:]-img[:,:(n-1)]!=0))
    return S/2/n**2
```

In [ ]:
```python
n = 64
betas = [0.65, 0.75, 0.85]
#tau = [[], [], []]
#Hs = [[], [], []]
# cdict = np.linspace(0,1,num=8)
for i in range(20):
    for b in range(3):
        beta = betas[b]
        wh = []
        bl = []
        white = np.ones((n,n), dtype=int)
        black = np.zeros((n,n), dtype=int)
        ww, bb = np.sum(white),np.sum(black)
        while ww != bb:
            swp(white, black, beta)
            ww, bb = np.sum(white),np.sum(black)
            wh.append(ww)
            bl.append(bb)
        # plt.imshow(white, cmap='gray')
        tau[b].append(len(wh))
        # sufficient statistics
        H = suff_stat(white)
        Hs[b].append(H)
```

In [ ]:
```python
print(tau, Hs)
```

In [ ]:
```python
means = [np.mean(x) for x in Hs]
means.append(-1)
taus = [np.mean(x) for x in tau]
print(means,taus)
```

In [ ]:

```
# cluster gibbs 1
def cluster_flip(img, beta):
    q = 1-np.exp(-beta)
    # clustering
    gid = 0
    cluster = np.zeros((n,n), dtype=int)
    for x in range(n):
        for y in range(n):
            if cluster[x,y]==0:
                gid += 1
                mark = np.copy(img[x,y])
                if np.random.randint(2)==1:
                    f = 1-mark
                else:
                    f = mark
                tree = [[x, y]]
                cluster[x,y] = 1
                i = 0
                while i < len(tree):
                    node = tree[i]
                    i += 1
                    # flip
                    x, y = node[0], node[1]
                    img[x, y] = f
                    # grow
                    pos = [[x+1, y],[x-1,y],[x,y+1],[x,y-1]]
                    for p in pos:
                        if p[0]>=0 and p[0]<n and p[1]>=0 and p[1]<n:
                            if cluster[p[0], p[1]]==0:
                                if mark==img[p[0], p[1]] and np.random.uniform(0,1) <= q:
                                        tree.append(p)
                                        cluster[p[0], p[1]] = 1
    return img, gid
```

In [ ]:

```
# cluster initialization
n = 64
const = np.ones((n,n), dtype=int)
a=np.tile([0,1],int(n/2))
b=np.tile([1,0],int(n/2))
checker = np.tile([a,b], [int(n/2),1]).astype(int)
betas = [0.65, 0.75, 0.85, 1]
```

In [ ]:
```
cts, cts1, cts2 = [], [], []
for b in range(4):
    beta = betas[b]
    C1 = np.copy(const)
    C2 = np.copy(checker)
    s1 = [suff_stat(C1)]
    s2 = [suff_stat(C2)]
    size = n**2
    sizes1, sizes2 = [], []
    for t in range(20):
        C1, gid = cluster_flip(C1, beta)
        sizes1.append(size/gid)
        C2, gid = cluster_flip(C2, beta)
        sizes2.append(size/gid)
        stat1 = suff_stat(C1)
        stat2 = suff_stat(C2)
        s1.append(suff_stat(C1))
        s2.append(suff_stat(C2))
    for i in range(20):
        if abs(s1[i]-means[b])<5e-3:
            cts1.append([i,s1[i]])
        if abs(s2[i]-means[b])<5e-3:
            cts2.append([i,s2[i]])
        if abs(s1[i]-s2[i])<5e-3:
            cts.append([i,s1[i]])
            break
    plt.close()
    plt.title("Version 1: Average cluster size: beta=%.2f"%beta)
    plt.xlabel('iteration')
    plt.ylabel('avg. size')
    plt.plot(sizes1, label='constant')
    plt.plot(sizes2, label='checkerboard')
    plt.legend()
    plt.savefig("1-3(%.2f).png"%beta)
    plt.close()
    plt.title("Version 1: Convergence plot: beta=%.2f"%beta)
    plt.xlabel('iteration')
    plt.ylabel('Sufficient Statistic')
    plt.plot(s1, label='constant')
    plt.plot(s2, label='checkerboard')
    plt.ylim(0,1)
    plt.plot(list(range(20)), np.repeat(means[b],20), c='k', linewidth=0.5)
    plt.legend()
    plt.savefig("1-1(%.2f).png"%beta)
```

In [ ]:
```
print(cts)
```

In [ ]:
```
print(cts1)
```

In [ ]:
```
print(cts2)
```

In [ ]:
```
def grow_flip(img, beta):
    n = img.shape[0]
    q = 1-np.exp(-beta)
    x, y = np.random.randint(n), np.random.randint(n)
    tree = [[x, y]]
    i = 0
    while i < len(tree):
        node = tree[i]
        i += 1
        # flip
        x, y = node[0], node[1]
        f = 1-img[x, y]
        img[x, y] = f
        # grow
        pos = [[x+1, y],[x-1,y],[x,y+1],[x,y-1]]
        for p in pos:
            if p not in tree and p[0]>=0 and p[0]<n and p[1]>=0 and p[1]<n:
                if img[x,y]!=img[p[0], p[1]] and np.random.uniform(0,1) <= q:
                        tree.append(p)
    count = len(tree)
    return img, count
```

In [ ]:
```
const = np.ones((n,n), dtype=int)
a=np.tile([0,1],int(n/2))
b=np.tile([1,0],int(n/2))
checker = np.tile([a,b], [int(n/2),1]).astype(int)
```

In [ ]:
```
#cts, cts1, cts2 = [], [], []
for b in range(3,4):
    beta = betas[b]
    C1 = np.copy(const)
    C2 = np.copy(checker)
    s1 = [suff_stat(C1)]
    s2 = [suff_stat(C2)]
    size = n**2
    sz1, sz2 = [], []
    count1, count2 = 0, 0
    itr1, itr2 = 0, 0
    i1, i2 = 1, 1
    for t in range(1000):
        C1, count = grow_flip(C1, beta)
        count1 += count
        if count1 > i1*size:
            i1 += 1
            stat1 = suff_stat(C1)
            s1.append(suff_stat(C1))
            sz1.append(size/(t-itr1))
            itr1 = t
        C2, count = grow_flip(C2, beta)
        count2 += count
        if count2 > i2*size:
            i2 += 1
            stat2 = suff_stat(C2)
            s2.append(suff_stat(C2))
            sz2.append(size/(t-itr2))
            itr2 = t
    print(len(s1), len(s2))
    for i in range(20):
        if abs(s1[i]-means[b])<5e-3:
            cts1.append([i,s1[i]])
        if abs(s2[i]-means[b])<5e-3:
            cts2.append([i,s2[i]])
        if abs(s1[i]-s2[i])<5e-3:
            cts.append([i,s1[i]])
    plt.close()
    plt.title("Version 2: Average cluster size: beta=%.2f"%beta)
    plt.xlabel('iteration')
    plt.ylabel('avg. size')
    plt.plot(sz1[:25], label='constant')
    plt.plot(sz2[:25], label='checkerboard')
    plt.legend()
    plt.savefig("2-3(%.2f).png"%beta)
    plt.close()
    plt.title("Version 2: Convergence plot: beta=%.2f"%beta)
    plt.xlabel('iteration')
    plt.ylabel('Sufficient Statistic')
    plt.plot(s1[:25], label='constant image')
    plt.plot(s2[:25], label='checkerboard image')
    plt.ylim(0,1)
    plt.plot(list(range(25)), np.repeat(means[b],25), c='k', linewidth=0.5)
    plt.legend()
    plt.savefig("2-1(%.2f).png"%beta)
```

In [ ]:
```
print(cts)
```

In [ ]:
```
print(cts1)
```

In [ ]:
```
print(cts2)
```