Stats202-HW2

Jiayu Wu | 905054229

Problem 1

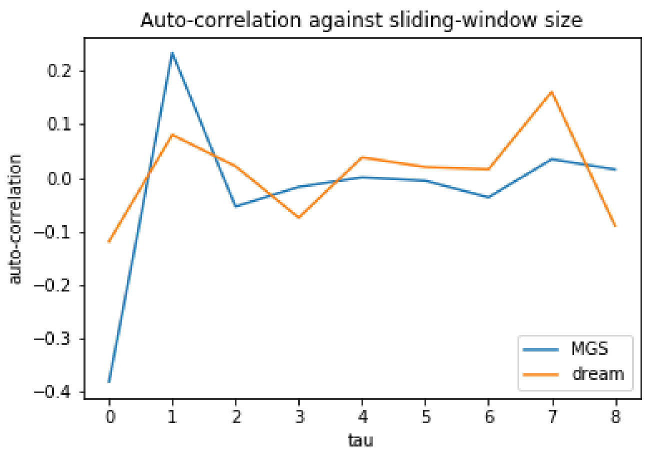
1. Edges $X_1 - X_2$ seems to be the bottleneck as the transition probablity is significantly larger. Edges $X_1 - X_2$ with $\kappa = 135635.35$.
$$\lambda_{slem} = 0.529 < 1 - \frac{1}{\kappa} = 0.9999926$$
2. $h = 0.3609$ with the subspace $S = \{X_3, X_4\}$
$$1 - 2h = 0.278 < \lambda_{slem} < 1 - \frac{h^2}{2} = 0.935$$
3. $h^* = 0.5157$ with the subspace $S = \{X_3, X_4\}$
$$1 - 2h^* = -0.0313 < \lambda_{slem} = 0 < 1 - \frac{h^{*2}}{2} = 0.867$$

Problem 2

$$K_{GMS} = \begin{bmatrix} 0. & 0.193 & 0.271 & 0.326 & 0.210 \\ 0.106 & 0.010 & 0.297 & 0.357 & 0.229 \\ 0.106 & 0.212 & 0.058 & 0.391 & 0.234 \\ 0.106 & 0.212 & 0.324 & 0.124 & 0.234 \\ 0.106 & 0.212 & 0.302 & 0.364 & 0.016 \end{bmatrix}$$

The off-diagonal elements are no less than that of K^*.

1. That of K_{GMS} is generally lower while the difference is not significant.



2. That of K_{GMS} is slightly closer to the truth, while both yields good extimates.
$$\theta_{MGS} = 12.408 \quad \theta^* = 12.254 \quad \theta_{true} = 12.452$$

Python Codes

```
In [18]: import numpy as np
         import matplotlib.pyplot as plt
         # %matplotlib inline
```

```
In [66]: # transition matrix and distribution
         K = np.array([[.2, .8, .0, .0, .0],
                       [.3, .0, .4, .0, .3],
                       [.1, .4, .0, .5, .0],
                       [.0, .0, .4, .2, .4],
                       [.0, .0, .3, .6, .1]])
         pi = K[0,]
         for i in range(100):
             pi = np.dot(pi,K)
         # lambda slem
         l_s = np.linalg.eigvals(K)[1]
         print(l_s)

         0.528764572617908
```

```
In [79]: pi

Out[79]: array([0.09613984, 0.17479971, 0.24471959, 0.29497451, 0.18936635])
```

```
In [80]: # bottleneck
         edges =  np.zeros((5,5))
         weights = 1/np.dot(np.diag(pi), K)
         states = list(range(5))
         nodes = []
         for x in range(5):
             nodes.append([n for n in states if K[x,n]!=0])

         for i in range(5):
             path = [['',i]]
             px = pi[i] # pi(x)
             w = 0 # accumulate gamma
             l = 0
             mask = [0]
             while l+1 == len(path):
                 x = path[l][1]
                 nds = nodes[x]
                 nds = [n for n in nds if [x,n] not in path]
                 if len(nds) > mask[l]:
                     y = nds[mask[l]]
                     w += weights[x,y]
                     path.append([x,y])
                     l += 1
                     mask.append(0)
         #        elif len(nodes) == 0:
         #            break
                 else:
                     if l == 0:
                         break
                     # if path[l][1] != path[0][1]:
                         # print(path)
                     for n in range(1,l+1):
                         # print(path[n], path[n+1])
                         edges[path[n][0], path[n][1]] += w*px*pi[path[l][1]]
                     w -= weights[path[l][0],path[l][1]]
                     del path[l]
                     del mask[l]
                     l -= 1
                     mask[l] += 1
         print(np.max(edges), 1-1/np.max(edges))
         l_s<1-1/np.max(edges)

         c:\users\fei960922\appdata\local\programs\python\python36\lib\site-packages\ipykernel_lau
         ncher.py:3: RuntimeWarning: divide by zero encountered in true_divide
           This is separate from the ipykernel package so we can avoid doing imports until

         135635.35373504678 0.9999926272909498

Out[80]: True
```

```
In [68]: # conductance
         def condunctance(K):
             states = list(range(5))
             S = [[[],0]] # subspaces, pi(S)'s, Q(S, Sc)'s
             Ks = np.dot(np.diag(pi), K) # pi(x)*K(x,y)
             l = 0
             while l < len(S):
                 s, m = S[l][0], S[l][1]
                 sc = [x for x in states if x not in s]
                 for n in sc:
                     m_new = m+pi[n]
                     if m_new <= 0.5:
                         new = [x for x in s]
                         new.append(n)
                         newc = [x for x in sc if x!=n]
                         Q = 0
                         for x in new:
                             for y in newc:
                                 Q += Ks[x,y]
                         if [new, m_new, Q] not in S:
                             S.append([new, m_new, Q])
                 l += 1

             Qs = [ss[2]/ss[1] for ss in S[1:] if ss[2]!=0]
             h = min(Qs)
             cl = S[np.argmin(Qs)+1][0]
             return h, cl
         h, cl = condunctance(K)
         print(cl, h, 1-2*h, 1-h**2/2)
         print((1-2*h)<=l_s, (1-h**2/2)>=l_s)
```

```
[3, 4] 0.3609022556390977 0.27819548872180455 0.9348747809373057
True True
```

```
In [71]: K_dream = np.tile(pi, 5).reshape((5,5))
         h_d, cl_d = condunctance(K_dream)
         print(cl_d, h_d, 1-2*h_d, 1-h_d**2/2)
         print((1-2*h_d)<=0,(1-h_d**2/2)>=0)
```

```
[4, 3] 0.5156591405680999 -0.03131828113619983 0.8670478253742843
True True
```

Problem 2

In [59]:
```
# K_MGS
K_new = K_dream.copy()
for i in range(5):
    K_new[i,i] = 0
    s = 1-K_dream[i,i]
    for j in range(5):
        K_new[i,j] = K_new[i,j]/s
# Q*alpha
eqn = np.dot(np.diag(pi), K_new)
for i in range(5):
    for j in range(i):
        if eqn[i,j]>eqn[j,i]:
            rej = K_new[i,j]*(1-eqn[j,i]/eqn[i,j])
            K_new[i,j] -= rej
            K_new[i,i] += rej
        elif eqn[i,j]<eqn[j,i]:
            rej = K_new[j,i]*(1-eqn[i,j]/eqn[j,i])
            K_new[j,i] -= rej
            K_new[j,j] += rej
print(K_new)
```

```
[[0.         0.19339243 0.2707494  0.32634972 0.20950846]
 [0.10636583 0.01013902 0.29655781 0.35745808 0.22947926]
 [0.10636583 0.21182701 0.05765462 0.39054966 0.23360288]
 [0.10636583 0.21182701 0.32401157 0.12419271 0.23360288]
 [0.10636583 0.21182701 0.30188679 0.3638814  0.01603896]]
```

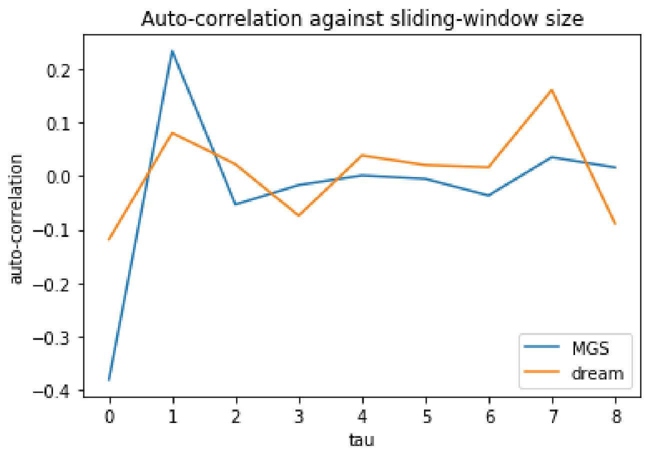
In [60]:
```
print(np.linalg.eigvals(K_new))
print(K_new>K_dream)
```

```
[-0.10636583  1.         -0.21756391 -0.20168799 -0.26635695]
[[False  True  True  True  True]
 [ True False  True  True  True]
 [ True  True False  True  True]
 [ True  True  True False  True]
 [ True  True  True  True False]]
```

In [74]:
```
# generate 500 samples
n = 500
x0 = 1
S1, S2 = [x0], [x0]
pi1, pi2 = np.zeros((1,5)),np.zeros((1,5))
for i in range(n):
    pr1,pr2 = K_new[S1[i]-1], K_dream[S2[i]-1]
    x1 = np.random.choice(5, p=pr1)
    x2 = np.random.choice(5, p=pr1)
    pi1[0,x1] += 1
    pi2[0,x2] += 1
    S1.append(x1+1)
    S2.append(x2+1)
```

In [75]:
```
# auto-correlation
taus = np.array(range(1,10))
mu1, mu2 = np.mean(S1[1:]), np.mean(S2[1:])
R1, R2 = [], []
for tau in taus:
    rho1, rho2 = 0, 0
    for t in range(1, n+1-tau):
        rho1 += (S1[t]-mu1)*(S1[t+tau]-mu1)
        rho2 += (S2[t]-mu2)*(S2[t+tau]-mu2)
    rho1 = rho1/(n-tau)
    rho2 = rho2/(n-tau)
    R1.append(rho1)
    R2.append(rho2)
```

In [76]:
```
plt.title('Auto-correlation against sliding-window size')
plt.xlabel('tau')
plt.ylabel('auto-correlation')
plt.plot(R1, label = 'MGS')
plt.plot(R2, label = 'dream')
plt.legend()
plt.savefig("1.png")
```



In [77]:
```
theta = np.dot(pi,np.array(range(1,6))**2)
pi1, pi2 = pi1/np.sum(pi1), pi2/np.sum(pi2)
theta1 = np.dot(pi1,np.array(range(1,6))**2)
theta2 = np.dot(pi2,np.array(range(1,6))**2)
print('MGS: %.3f'%theta1, 'K_dream: %.3f'%theta2, 'Expectation: %.3f'%theta)
```

```
MGS: 12.408 K_dream: 12.254 Expectation: 12.452
```