Assignment 2

Abdulkadir Çelikkanat

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import linalg as LA
```

Problem 1 (Importance Sampling)

Let R be $\{(x,y) : |x| + |y| \leq 1\}$ which is rotated square as in the previous assignment, and $\Omega = \{(x,y) | x^2 + y^2 \leq 1\}$ for p=0.5, and say that area of the region Ω is c. Then, define the function $f(x) = c\mathbb{1}_{\Omega}$. Therefore, the expectation of f(x)

$$\mathbb{E}_p[f(x)] = \int_R f(x)dp = \int_{\Omega} c\mathbb{1}_{\Omega} dp = cp(\Omega) = c$$

gives the area of Ω, where p is the uniform probability measure function on Ω. After changing the probabiliy measure in the equation, it can be rewritten as

$$\mathbb{E}_p[f(x)] = \int_{\Omega} f(x)dp = \int_R f(x)\frac{p(x)}{q(x)}dq = \int_R f(x)w(x)dq = \mathbb{E}_q[f(x)w(x)]$$

q(x) is uniform probability measure on R. Then, by Monte Carlo estimation, it can be obtained that

$$c = \mathbb{E}_p[f(x)] = \frac{1}{N}\sum_{i=1}^{N} f(x^{(i)})w(x^{(i)}) = \frac{1}{N}\sum_{i=1}^{N} c\mathbb{1}_{\Omega}(x^{(i)})\frac{\frac{1}{c}}{\frac{1}{2}} = \frac{2}{N}\sum_{i=1}^{N} \mathbb{1}_{\Omega}(x^{(i)})$$

where $x^{(i)} \sim$ q(x). The last equation indeed equals to rejection sampling

Firstly, the function to uniformly sample points from the region **R** is implemented below.

```
def sampleFromRotatedSquare(N):
# Definiton of rotation matrix in counter clockwise direction
R = np.array([[np.sqrt(2.0)/2, -np.sqrt(2.0)/2], [np.sqrt(2.0)/2, np.sqrt(2.0)/2]])

# Draw random points from the square [-2^(0.5)/2, 2^(0.5)/2]^2
x = np.random.uniform(-np.sqrt(2.0)/2, np.sqrt(2.0)/2, N)
y = np.random.uniform(-np.sqrt(2.0)/2, np.sqrt(2.0)/2, N)

# Rotate points
[rx, ry] = R.dot([x, y])

return np.vstack((rx, ry))
```

Now the area of the region Ω can be found as described above

```python
N = 1000 #number of samples
samples = sampleFromRotatedSquare(N)

p = 0.5
# Get the samples inside
inx = LA.norm(samples, ord=p, axis=0) < 1

c = (2.0/N) * (np.sum(inx))
print "The estimated area : " + str(c)
```

The estimated area : 0.66558
...which is close to the real value $\frac{2}{3} = 0.\tilde{6}$
Since the variance of f(x)w(x) equals to

$$Var_q(f(x)w(x)) = Var_q(f(x)\frac{p(x)}{q(x)}) = Var_q(c\mathbb{1}_\Omega \frac{1/c}{1/2}) = 4Var_q(\mathbb{1}_\Omega)$$

```python
var = 4.0*np.var(inx)
print "Variance : " + str(var)
```

Variance : 0.8775

Problem 2

a) Construction of the transition kernel

```python
A = np.array([[0.25, 0, 0, 0, 0, 0.25],
[0.25, 0.25, 0.25, 0.25, 0.25, 0.25],
[0.25, 0.25, 0, 0.5, 0.25, 0.25],
[0, 0, 0.5, 0, 0, 0],
[0, 0.25, 0, 0, 0.25, 0],
[0.25, 0.25, 0.25, 0.25, 0.25, 0.25]])
print A
```

b) Stationary distribution

Since $A^3 > 0$, the transition matrix A is regular, so ergodic.

```python
(D, Q) = np.linalg.eig(A)
# Get the first eigenvector which corresponds to eigenvalue 1
u = Q[:,0]
# Normalize eigenvector
u = u / np.sum(u)
print "Eigenvalues : " + str(np.around(D, decimals=4))
print "Stationary distribution"
print u
```

Eigenvalues : [1. 0.25 0. -0.5 0.25 0.]
Stationary distribution [0.0833 0.25 0.222 0.111 0.0833 0.25] .., so the stationary distribution is not uniform.

c) Detailed balance equation

To satisfy the detailed balance equation, the matrix obtained by elementwise multiplication of A with eigenvector 'u' should be symmetric

```
# Multiply elementwise the kernel 'A' and eigenvector 'u'
S = np.multiply(A,u)
# Check whether 'S' is symmetric or not
print np.allclose(S.T, S)
```

False
Hence, it does not satify the detailed balance equation.

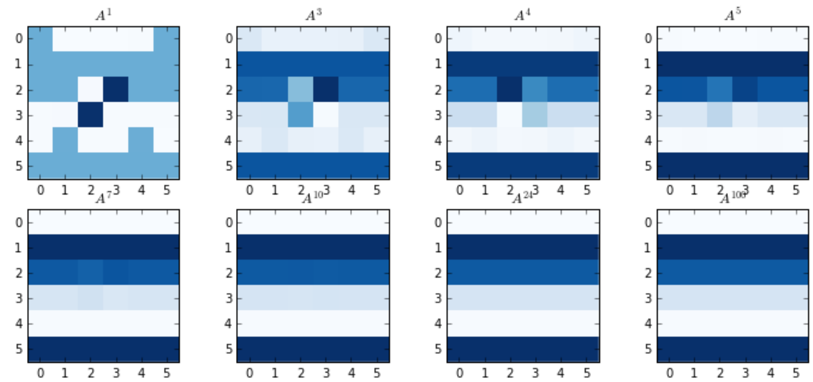
d) The mixing time

```
epsilon = (1e-8)

Tmix = 1
p = [1, 0, 0, 0, 0, 0] #Initial position
s = np.dot(A,p)
while (0.5*LA.norm(s-u, ord=1) > epsilon):
s = np.dot(A,s)
Tmix += 1

print "The mixing time : " + str(Tmix)
```

The mixing time : 24

e) Visualization of some intermediary powers of A

```
(f, axs) = plt.subplots(2,4,figsize=(12,5))

for inx, power in enumerate([1,3,4,5,7,10,24,100]):
E = LA.matrix_power(A,power)
im = axs[inx/4][inx%4].imshow(E, interpolation='none',
    cmap='Blues')
axs[inx/4][inx%4].set_title('$A^{'+str(power)+'}$')
plt.show()
```

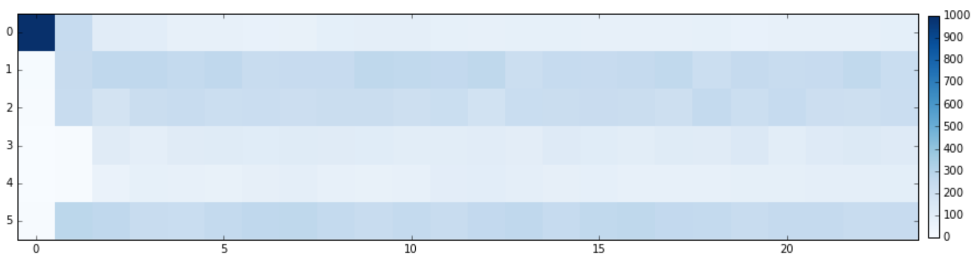


f) Independent Chains

The function below is used to generate chains for a given transiton kernel K, initial state, number of chains N, and length of chain M

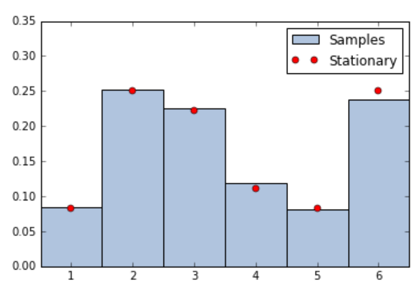
```
def generateChains(K, initState, N, M):
# K is the transition kernel
# N is the number of chains
# M is the length of chain
chains = np.zeros((N,M), dtype=np.int)
chains[:,0] = initState-1 # -1, since index starts from 0
for n in range(N):
for m in range(1,M):
chains[n][m] = np.random.choice(range(6), p=K[:,chains[n][m-1]])

return (chains+1) # +1, since index starts from 0
```

```
# Generate 1000 independent sequence of length Tmix with inital state '1'
chains = generateChains(A,1,1000,Tmix)

# Generate histograms
col = np.zeros(1000)
t = np.array([[j for j in np.histogram(chains[:,i], bins=range(1,8))[0][:]] for i in range(Tmix)])

# Plot the figure
fig, ax = plt.subplots(figsize=(15, 5))
im = ax.imshow(t.T,interpolation='none', cmap='Blues')
fig.colorbar(im, fraction=0.012, pad=0.01)
plt.show()
```



g) Dependent Samples

```
# Generate a chain of length 1000
chain = generateChains(A,1,1,1000)[0]

# Extract the histogram after discarding first Tmix terms
(counts, edges) = np.histogram(chain[Tmix::], bins=range(1,8),density=True)

# Plot the histogram and stationary distribution
xaxis = np.arange(1,7) + 0.5
p1 = plt.bar(range(1,7), counts, color='LightSteelBlue', width=1.0)
p2 = plt.plot(xaxis,u,'ro')
plt.xticks(xaxis, range(1,7))
plt.ylim([0, 0.35])
plt.legend((p1[0], p2[0]), ('Samples', 'Stationary'))
plt.show()
```




```
print "Stationary distribution"
print np.around(u,decimals=4)
print "Sample distribution"
print np.around(counts,decimals=4)
```

Stationary distribution
0.0833 0.25 0.2222 0.1111 0.0833 0.25

Sample distribution
0.085 0.252 0.2254 0.1189 0.0809 0.2377