EECS 545 – Machine Learning - Homework #4

Mina Jafari
group member: Joshua Kammeraad

1) **Question 1.**
(a)

$$
\begin{aligned}
D_{KL}(p||q) &= \sum_{x\in X}\sum_{y\in Y} p(x,y)\log\frac{p(x,y)}{q(x,y)} \\
&= \sum_{x\in X}\sum_{y\in Y} p(x,y)\log p(x,y) - \sum_{x\in X}\sum_{y\in Y} p(x,y)\log q(x,y) \\
&= \sum_{x\in X}\sum_{y\in Y} p(x,y)\log p(x,y) - \sum_{x\in X}\sum_{y\in Y} p(x,y)\log\left(q_1(x)q_2(y)\right) \\
&= \sum_{x\in X}\sum_{y\in Y} p(x,y)\log p(x,y) - \sum_{x\in X}\sum_{y\in Y} p(x,y)\log q_1(x) - \sum_{x\in X}\sum_{y\in Y} p(x,y)\log q_2(y) \\
&= \sum_{x\in X}\sum_{y\in Y} p(x,y)\log p(x,y) - \sum_{x\in X} p(x)\log q_1(x) - \sum_{y\in Y} p(y)\log q_2(y) \geq 0
\end{aligned}
$$

The first term is constant w.r.t. $q_1(x)$ and $q_2(y)$. So, we should only minimize the q terms

$$-\sum_{x\in X} p(x)\log q_1(x) - \sum_{y\in Y} p(y)\log q_2(y)$$

Minimizing this term is equivalent to minimizing

$$\sum_{x\in X} p(x)\log p(x) + \sum_{y\in Y} p(y)\log p(y) - \sum_{x\in X} p(x)\log q_1(x) - \sum_{y\in Y} p(y)\log q_2(y)$$

since the first two terms are constsnt as a function of q.

The KL-divergence is minimized when it equals 0

$\implies q_1(x) = p(x)$ and $q_2(y) = p(y)$

(b)

$$
\begin{aligned}
D_{KL}(q||p) &= \sum_{x\in X}\sum_{y\in Y} q(x,y)\log\frac{q(x,y)}{p(x,y)} \\
&= \sum_{x\in X}\sum_{y\in Y} q_1(x)q_2(y)\log\frac{q_1(x)q_2(y)}{p(x,y)} \\
&= \sum_{x\in X}\sum_{y\in Y} q_1(x)q_2(y)\log\left(q_1(x)q_2(y)\right) - \sum_{x\in X}\sum_{y\in Y} q_1(x)q_2(y)\log p(x,y) \\
&= \sum_{x\in X}\sum_{y\in Y} q_1(x)q_2(y)\log q_1(x) + \sum_{x\in X}\sum_{y\in Y} q_1(x)q_2(y)\log q_2(y) \\
&\quad - \sum_{x\in X}\sum_{y\in Y} q_1(x)q_2(y)\log p(x,y) \\
\mathcal{L}(q_1(x),q_2(y),\lambda_1,\lambda_2) &= \sum_{x\in X}\sum_{y\in Y} q_1(x)q_2(y)\log q_1(x) + \sum_{x\in X}\sum_{y\in Y} q_1(x)q_2(y)\log q_2(y) \\
&\quad - \sum_{x\in X}\sum_{y\in Y} q_1(x)q_2(y)\log p(x,y) \\
&\quad + \lambda_1\left(\sum_{x\in X} q_1(x) - 1\right) + \lambda_2\left(\sum_{y\in Y} q_2(y) - 1\right) \\
&= \sum_{x\in X} q_1(x)\log q_1(x) + \sum_{y\in Y} q_2(y)\log q_2(y) - \sum_{x\in X}\sum_{y\in Y} q_1(x)q_2(y)\log p(x,y) \\
&\quad + \lambda_1\left(\sum_{x\in X} q_1(x) - 1\right) + \lambda_2\left(\sum_{y\in Y} q_2(y) - 1\right) \\
\frac{\partial\mathcal{L}}{\partial q_1(x_i)} &= \log q_1(x_i) + 1 - \sum_{y\in Y} q_2(y)\log p(x_i,y) + \lambda_1 = 0 \\
\frac{\partial\mathcal{L}}{\partial q_2(y_j)} &= \log q_2(y_j) + 1 - \sum_{x\in X} q_1(x)\log p(x,y_j) + \lambda_2 = 0
\end{aligned}
$$

$$\frac{\partial \mathcal{L}}{\partial q_1(x_1)} = \log q_1(x_1) + 1 - q_2(y_1)\log p(x_1, y_1) - q_2(y_2)\log p(x_1, y_2) + \lambda_1 = 0$$
$$\frac{\partial \mathcal{L}}{\partial q_1(x_2)} = \log q_1(x_2) + 1 - q_2(y_1)\log p(x_2, y_1) - q_2(y_2)\log p(x_2, y_2) + \lambda_1 = 0$$
$$\implies q_1(x_1) = q_1(x_2)$$
$$\frac{\partial \mathcal{L}}{\partial q_1(x_3)} = \log q_1(x_3) + 1 - q_2(y_3)\log p(x_3, y_3) + \lambda_1 = 0$$
$$\frac{\partial \mathcal{L}}{\partial q_1(x_4)} = \log q_1(x_4) + 1 - q_2(y_4)\log p(x_4, y_4) + \lambda_1 = 0$$
$$\frac{\partial \mathcal{L}}{\partial q_2(y_1)} = \log q_2(y_1) + 1 - q_1(x_1)\log p(x_1, y_1) - q_1(x_2)\log p(x_2, y_1) + \lambda_2 = 0$$
$$\frac{\partial \mathcal{L}}{\partial q_2(y_2)} = \log q_2(y_2) + 1 - q_1(x_1)\log p(x_1, y_2) - q_1(x_2)\log p(x_2, y_2) + \lambda_2 = 0$$
$$\implies q_2(y_1) = q_2(y_2)$$
$$\frac{\partial \mathcal{L}}{\partial q_2(y_3)} = \log q_2(y_3) + 1 - q_1(x_3)\log p(x_3, y_3) + \lambda_2 = 0$$
$$\frac{\partial \mathcal{L}}{\partial q_2(y_4)} = \log q_2(y_4) + 1 - q_1(x_4)\log p(x_4, y_4) + \lambda_2 = 0$$

$q_1(x_3)q_2(y_1) = 0$
$q_1(x_3)q_2(y_2) = 0$
$q_1(x_3)q_2(y_3) = 0$
if $q_1(x_3) \neq 0 \implies q_2(y_3) = 1$
$q_2(y_3)q_1(x_1) = 0$
$q_2(y_3)q_1(x_2) = 0$
$q_2(y_3)q_1(x_4) = 0$
if $q_2(y_3) \neq 0 \implies q_1(x_3) = 1$
$q_1(x_4)q_2(y_1) = 0$
$q_1(x_4)q_2(y_2) = 0$
$q_1(x_4)q_2(y_3) = 0$
if $q_1(x_4) \neq 0 \implies q_2(y_4) = 1$
$q_2(y_4)q_1(x_1) = 0$
$q_2(y_4)q_1(x_2) = 0$
$q_2(y_4)q_1(x_3) = 0$
if $q_2(y_4) \neq 0 \implies q_1(x_4) = 1$
$q_1(x_1)q_2(y_3) = 0$
$q_1(x_1)q_2(y_4) = 0$
$q_1(x_2)q_2(y_3) = 0$
$q_1(x_2)q_2(y_4) = 0$
$\implies q_1(x_1) = q_1(x_2) = 0.5$ and $q_2(y_1) = q_2(y_2) = 0.5$

minimum 1: $x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0, y_1 = 0, y_2 = 0, y_3 = 1, y_4 = 0, D_{KL}(q||p) = 2$
minimum 2: $x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 1, y_1 = 0, y_2 = 0, y_3 = 0, y_4 = 1, D_{KL}(q||p) = 2$
minimum 3: $x_1 = 0.5, x_2 = 0.5, x_3 = 0, x_4 = 0, y_1 = 0.5, y_2 = 0.5, y_3 = 0, y_4 = 0, D_{KL}(q||p) = 1$

(c)

$$
\begin{aligned}
D_{KL}(q||p) &= \sum_{x \in X} \sum_{y \in Y} p(x)p(y) \log \frac{p(x)p(y)}{p(x,y)} \\
&= \cdots + p(x_1)p(y_3) \log \left(\frac{p(x_1)p(y_3)}{0} \right) + \cdots \\
&= \cdots + \frac{1}{4}\frac{1}{4} \log \left(\frac{\frac{1}{4}\frac{1}{4}}{0} \right) + \cdots = \cdots + \frac{1}{16} \log (\infty) + \cdots = \infty
\end{aligned}
$$

2) **Question 2.**

(a)

Since Σ is positive definite:

$$p(x_1, x_2) = \frac{1}{2\pi\sqrt{|\Sigma|}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu)\right]$$

$$p(x_1) = \frac{1}{\sqrt{2\pi|\Sigma_{11}|}} \exp\left[-\frac{1}{2}(x_1-\mu_1)^\top \Sigma_{11}^{-1}(x_1-\mu_1)\right]$$

$$p(x_2) = \frac{1}{\sqrt{2\pi|\Sigma_{22}|}} \exp\left[-\frac{1}{2}(x_2-\mu_2)^\top \Sigma_{22}^{-1}(x_2-\mu_2)\right]$$

$$A = \Sigma^{-1}$$

$$\begin{aligned}
p(x_1|x_2) &= \frac{p(x_1,x_2)}{p(x_2)} \\
&= \frac{1}{2\pi\sqrt{|\Sigma|}} \exp\Bigg[-\frac{1}{2}\big((x_1-\mu_1)A_{11}(x_1-\mu_1) + (x_1-\mu_1)A_{12}(x_2-\mu_2) \\
&+ (x_2-\mu_2)A_{21}(x_1-\mu_1) + (x_2-\mu_2)A_{22}(x_2-\mu_2)\big)\Bigg] \div \\
&\frac{1}{\sqrt{2\pi|\Sigma_{22}|}} \exp\left[-\frac{1}{2}(x_2-\mu_2)^\top \Sigma_{22}^{-1}(x_2-\mu_2)\right]
\end{aligned}$$

$$\begin{aligned}
p(x_1|x_2) &= \sqrt{\frac{|\Sigma_{22}|}{2\pi|\Sigma|}} \exp\Bigg[-\frac{1}{2}\big((x_1-\mu_1)A_{11}(x_1-\mu_1) + (x_1-\mu_1)A_{12}(x_2-\mu_2) \\
&+ (x_2-\mu_2)A_{21}(x_1-\mu_1) + (x_2-\mu_2)A_{22}(x_2-\mu_2) - (x_2-\mu_2)^\top \Sigma_{22}^{-1}(x_2-\mu_2)\big)\Bigg]
\end{aligned}$$

$$\begin{aligned}
p(x_2|x_1) &= \frac{p(x_1,x_2)}{p(x_1)} \\
&= \sqrt{\frac{|\Sigma_{11}|}{2\pi|\Sigma|}} \exp\Bigg[-\frac{1}{2}\big((x_1-\mu_1)A_{11}(x_1-\mu_1) + (x_1-\mu_1)A_{12}(x_2-\mu_2) \\
&+ (x_2-\mu_2)A_{21}(x_1-\mu_1) + (x_2-\mu_2)A_{22}(x_2-\mu_2) - (x_1-\mu_1)^\top \Sigma_{11}^{-1}(x_1-\mu_1)\big)\Bigg]
\end{aligned}$$

$$\begin{aligned}
p(x_1|x_2) &= \sqrt{\frac{2}{3\pi}} \exp\left[-\frac{1}{2}\left(\frac{4}{3}(x_1-1)^2 + \frac{1}{3}(x_2-1)^2 - \frac{4}{3}(x_1-1)(x_2-1)\right)\right] \\
&= \sqrt{\frac{2}{3\pi}} \exp\left[-\frac{1}{6}(2x_1-x_2-1)^2\right] \\
p(x_2|x_1) &= \sqrt{\frac{2}{3\pi}} \exp\left[-\frac{1}{2}\left(\frac{1}{3}(x_1-1)^2 + \frac{4}{3}(x_2-1)^2 - \frac{4}{3}(x_1-1)(x_2-1)\right)\right] \\
&= \sqrt{\frac{2}{3\pi}} \exp\left[-\frac{1}{6}(2x_2-x_1-1)^2\right]
\end{aligned}$$

$$\sigma_{x_1}^2 = \sigma_{x_2}^2 = \frac{3}{4} \ , \ \mu_{x_1} = 0.5x_2 + 0.5 \ , \ \mu_{x_2} = 0.5x_1 + 0.5$$

(b)

Figure 1: $p(x_1)$

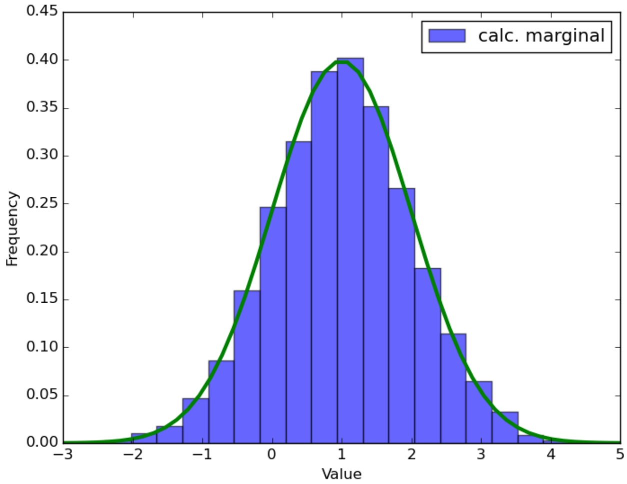
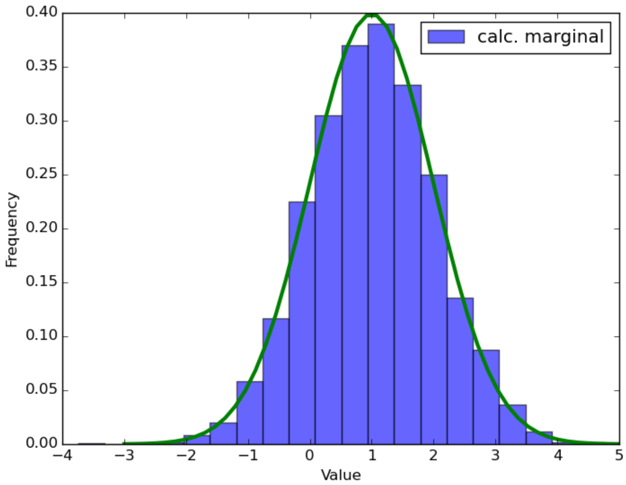


Figure 2: $p(x_2)$




```
import numpy as np
import matplotlib.pyplot as plt
from math import pi
import matplotlib.mlab as mlab

def updateX1(x2):
    sigma = np.sqrt(3.0/4.0)
    mu = 0.5*x2+0.5
    x1 = sigma * np.random.randn() + mu
    return x1


def updateX2(x1):
    sigma = np.sqrt(3.0/4.0)
    mu = 0.5*x1+0.5
    x2 = sigma * np.random.randn() + mu
    return x2

def main():
    x1 = 0.0
    N = 5000
    x1Values = np.zeros(N)
    x2Values = np.zeros(N)

    for i in range(N):
        x2 = updateX2(x1)
        x2Values[i] = x2
        x1 = updateX1(x2)
        x1Values[i] = x1
```

```
    #plot
    plt.hist(x1Values, bins=20, alpha=0.6, label='calc. marginal', normed=True)
    x = np.linspace(-3,5)
    plt.plot(x,mlab.normpdf(x,1.0,1.0), lw=3)
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    plt.legend(loc='upper right')
    plt.savefig('px1')
    plt.clf()

    plt.hist(x2Values, bins=20, alpha=0.6, label='calc. marginal', normed=True)
    x = np.linspace(-3,5)
    plt.plot(x,mlab.normpdf(x,1.0,1.0), lw=3)
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    plt.legend(loc='upper right')
    plt.savefig('px2')

if __name__ == "__main__":
    main()
```

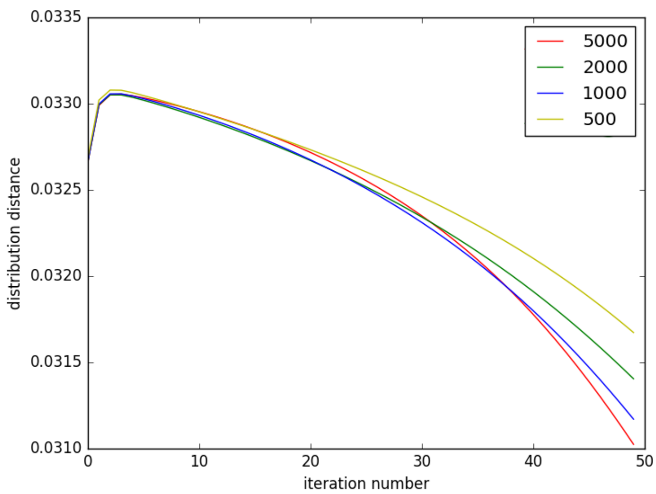
3) **Question 3.**

(a)

Most Probable State Sequences	Prior Probability	Likelihood	Posterior Probability
0222	0.0375	0.1000	0.0007
0122	0.0200	0.1800	0.0007
0201	0.0120	0.2880	0.0007

```
import numpy as np
import itertools

A = np.array([[0.5, 0.2, 0.3], [0.2, 0.4, 0.4], [0.4, 0.1, 0.5]])
phi = np.array([[0.8, 0.2], [0.1, 0.9], [0.5, 0.5]])
pi0 = np.array([0.5, 0.3, 0.2])

combinations = list(itertools.product([0,1,2], repeat=4))
output = [[0 for x in range(4)] for x in range(81)]
i = 0
for elem in combinations:
    output[i][0] = elem #combinations
    output[i][1] = (pi0[elem[0]] * A[elem[0]][elem[1]] * A[elem[1]][elem[2]] * A[elem[2]][elem[3]]) #prior
    output[i][2] = (phi[elem[0]][0] * phi[elem[1]][1] * phi[elem[2]][0] * phi[elem[3]][1]) #likelihood
    i += 1

colSum = 0
for i in range(len(output)):
    colSum += output[i][2]

for i in range(len(output)):
    output[i][3] = output[i][1] * output[i][2] / colSum

output = sorted(output, key=lambda row:row[3], reverse=True)

for col1, col2, col3, col4 in output:
        print (col1, "%.4f" % col2, "%.4f" % col3, "%.4f" % col4)
```

(b)




```
from __future__ import division
import numpy as np
from sklearn.preprocessing import normalize
from matplotlib import pyplot as plt
from copy import deepcopy

# 3 states, 4 observations
# Generate the data according to the specification in the homework description
# for part (b)
A = np.array([[0.5, 0.2, 0.3], [0.2, 0.4, 0.4], [0.4, 0.1, 0.5]])
phi = np.array([[0.8, 0.2], [0.1, 0.9], [0.5, 0.5]])
pi = np.array([0.5, 0.3, 0.2])

XData = []

for _ in xrange(5000):
```

```
    z = [np.random.choice([0,1,2], p=pi)]
    for _ in range(3):
        z.append(np.random.choice([0,1,2], p=A[z[-1]]))
    x = [np.random.choice([0,1], p=phi[zi]) for zi in z]
    XData.append(x)

# TODO: Implement Baum-Welch for estimating the parameters of the HMM

def fb_algorithm(A_mat, B_mat, pi_mat, observation):
    k = len(observation)
    alpha = np.zeros((3, k))
    beta = np.zeros((3, k))

    # setting initial alpha
    alpha[0, 0] = pi_mat[0] * B_mat[0, observation[0]]
    alpha[1, 0] = pi_mat[1] * B_mat[1, observation[0]]
    alpha[2, 0] = pi_mat[2] * B_mat[2, observation[0]]
    # updating alpha
    for i in range(1, k):
        alpha[0, i] = B_mat[0][observation[i]] * (alpha[0][i-1] * A_mat[0][0] \
                      + alpha[1][i-1] * A_mat[1][0] + alpha[2][i-1] * A_mat[2][0])
        alpha[1, i] = B_mat[1][observation[i]] * (alpha[0][i-1] * A_mat[0][1] \
                      + alpha[1][i-1] * A_mat[1][1] + alpha[2][i-1] * A_mat[2][1])
        alpha[2, i] = B_mat[2][observation[i]] * (alpha[0][i-1] * A_mat[0][2] \
                      + alpha[1][i-1] * A_mat[1][2] + alpha[2][i-1] * A_mat[2][2])

    # setting initial beta
    beta[:, -1] = 1.0
    # updating beta
    for i in range(k-2, -1, -1):
        beta[0, i] = (beta[0][i+1] * A_mat[0][0] * B_mat[0][observation[i+1]]+ \
                      beta[1][i+1] * A_mat[0][1] * B_mat[1][observation[i+1]]+\
                      beta[2][i+1] * A_mat[0][2] * B_mat[2][observation[i+1]])
        beta[1, i] = (beta[0][i+1] * A_mat[1][0] * B_mat[0][observation[i+1]]+\
                      beta[1][i+1] * A_mat[1][1] * B_mat[1][observation[i+1]]+\
                      beta[2][i+1] * A_mat[1][2] * B_mat[2][observation[i+1]])
        beta[2, i] = (beta[0][i+1] * A_mat[2][0] * B_mat[0][observation[i+1]]+\
                      beta[1][i+1] * A_mat[2][1] * B_mat[1][observation[i+1]]+\
                      beta[2][i+1] * A_mat[2][2] * B_mat[2][observation[i+1]])

    gamma = alpha * beta
    gamma = gamma / np.sum(gamma, axis=0)

    ksi = np.zeros((3,3,3))
    for i in range(3):
        for j in range(3):
            for k in range(3):
                ksi[i, j, k] = alpha[i][k] * A_mat[i][j] * beta[j][k+1] * B_mat[j][observation[k+1]]\
                               / np.sum(alpha[:, -1])

    return gamma, ksi


def baumWelch(A_mat, B_mat, pi_mat, obs_mat):
    # updating pi
    newPi = np.zeros((3))
    numPi_1 = numPi_2 = numPi_3 = 0.0
    denumPi = 0.0
    numB_00 = numB_10 = numB_20 = numB_01 = numB_11 = numB_21 = 0.0
    denumB_0 = denumB_1 = denumB_2 = 0.0
    newB = np.zeros((3, 2))

    numOfDataPts = len(obs_mat)
    for i in range(numOfDataPts):
        gamma, ksi, pOfX = fb_algorithm(A_mat, B_mat, pi_mat, obs_mat[i])
        numPi_1 += np.sum(gamma[0, :])
        numPi_2 += np.sum(gamma[1, :])
        numPi_3 += np.sum(gamma[2, :])
        denumPi += (np.sum(gamma[0, :]) + np.sum(gamma[1, :]) + np.sum(gamma[2, :]))

        for j in range(gamma.shape[1]):
            numB_00 += (obs_mat[i][j] == 0) * gamma[0, j]
            numB_01 += (obs_mat[i][j] == 1) * gamma[0, j]
            numB_10 += (obs_mat[i][j] == 0) * gamma[1, j]
            numB_11 += (obs_mat[i][j] == 1) * gamma[1, j]
            numB_20 += (obs_mat[i][j] == 0) * gamma[2, j]
            numB_21 += (obs_mat[i][j] == 1) * gamma[2, j]
        denumB_0 += np.sum(gamma[0, :])
        denumB_1 += np.sum(gamma[1, :])
        denumB_2 += np.sum(gamma[2, :])
    newPi[0] = numPi_1 / denumPi
    newPi[1] = numPi_2 / denumPi
```

```
    newPi[2] = numPi_3 / denumPi
    # updating B
    newB[0, 0] = numB_00 / denumB_0
    newB[0, 1] = numB_01 / denumB_0
    newB[1, 0] = numB_10 / denumB_1
    newB[1, 1] = numB_11 / denumB_1
    newB[2, 0] = numB_20 / denumB_2
    newB[2, 1] = numB_21 / denumB_2

    # updating A
    newA = np.zeros((3, 3))
    for j in range(ksi.shape[0]):
        for k in range(ksi.shape[1]):
            num_jk = denum_jk = 0.0
            for i in range(numOfDataPts):
                gamma, ksi, pOfX = fb_algorithm(A_mat, B_mat, pi_mat, obs_mat[i])
                num_jk += np.sum(ksi[j, k, :])
                denum_jk += np.sum(ksi[j, :, :])
            newA[j, k] = num_jk / denum_jk

    return newPi, newA, newB

def main():

    N = [500, 1000, 2000, 5000]

    A0 = np.random.uniform(0,1,9).reshape(3, 3)
    A0 = normalize(A0, axis=1, norm='l1')
    B0 = np.random.uniform(0,1,6).reshape(3, 2)
    B0 = normalize(B0, axis=1, norm='l1')
    pi0 = np.random.uniform(0,1,3).reshape(3, 1)
    pi0 = normalize(pi0, axis=1, norm='l1')

    iterNum = 50
    calcDist_500 = []
    calcDist_1000 = []
    calcDist_2000 = []
    calcDist_5000 = []
    ##
    import itertools
    combinations = list(itertools.product([0,1], repeat=4))
    ##
    for n in N:
        counter = 0
        xDataInput = XData[0:n]
        ##
        outTrue = [[0 for x in range(4)] for x in range(16)]
        i = 0
        for elem in combinations:
                outTrue[i][1] = (pi[elem[0]] * A[elem[0]][elem[1]] * A[elem[1]][elem[2]] * A[elem[2]][elem[3]])
                outTrue[i][2] = (phi[elem[0]][0] * phi[elem[1]][1] * phi[elem[2]][0] * phi[elem[3]][1])
                i += 1
        colSum = np.sum(outTrue[:][2])
        for i in range(len(outTrue)):
            outTrue[i][3] = outTrue[i][1] * outTrue[i][2] / colSum
        ##
        piCur, ACur, BCur = baumWelch(A0, B0, pi0, XData[0:n])
        while (counter < iterNum):
            piOld = deepcopy(piCur)
            AOld = deepcopy(ACur)
            BOld = deepcopy(BCur)
            ##
            output = [[0 for x in range(4)] for x in range(16)]
            i = 0
            for elem in combinations:
                output[i][1] = (piOld[elem[0]] * AOld[elem[0]][elem[1]] * AOld[elem[1]][elem[2]] * AOld[elem[2]]
                output[i][2] = (BOld[elem[0]][0] * BOld[elem[1]][1] * BOld[elem[2]][0] * BOld[elem[3]][1])
                i += 1
            colSum = np.sum(output[:][2])
            for i in range(len(output)):
                output[i][3] = output[i][1] * output[i][2] / colSum
            ##
            piCur, ACur, BCur = baumWelch(AOld, BOld, piOld, xDataInput)
            diff = 0.5 * np.sum(np.absolute([a - b for a, b in zip(output[:][3], outTrue[:][3])]))
            if n == 500:
                calcDist_500.append(diff)
            elif n == 1000:
                calcDist_1000.append(diff)
            elif n == 2000:
                calcDist_2000.append(diff)
            elif n == 5000:
                calcDist_5000.append(diff)
```

```
            counter += 1

    print ACur
    print piCur
    print BCur

    xAxis = range(iterNum)
    plt.plot(xAxis, calcDist_500, 'r', label='500')
    plt.plot(xAxis, calcDist_1000, 'g', label='1000')
    plt.plot(xAxis, calcDist_2000, 'b', label='2000')
    plt.plot(xAxis, calcDist_5000, 'y', label='5000')
    plt.ylabel('distribution distance')
    plt.xlabel('iteration number')
    plt.legend()
    plt.savefig('plot-b.png')

if __name__ == "__main__":
    main()
```

4) **Question 4.**

```python
import numpy as np
from matplotlib import pyplot
import matplotlib as mpl
from sklearn.svm import SVR


def show_image(image):
    """
    Render a given numpy.uint8 2D array of pixel data.
    """
    fig = pyplot.figure()
    ax = fig.add_subplot(1,1,1)
    imgplot = ax.imshow(image, cmap=mpl.cm.Greys)
    imgplot.set_interpolation('nearest')
    ax.xaxis.set_ticks_position('top')
    ax.yaxis.set_ticks_position('left')
    pyplot.show()

def get_patches(X):
    m,n = X.shape
    X = np.pad(X, ((2, 2), (2, 2)), 'constant')
    patches = np.zeros((m*n, 25))
    for i in range(m):
        for j in range(n):
            patches[i*n+j] = X[i:i+5,j:j+5].reshape(25)
    return patches

# input
A = np.loadtxt("train_noised.csv", delimiter=",", skiprows=1)
A = A / 255.0
B = np.loadtxt("train_clean.csv", delimiter=",", skiprows=1)
B = B / 255.0
C = np.loadtxt("test_noised.csv", delimiter=",", skiprows=1)
C = C / 255.0
trainData = np.delete(A, 0, 1)
target = np.delete(B, 0, 1)
test = np.delete(C, 0, 1)


def get_patches(X):
    m,n = X.shape
    X = np.pad(X, ((2, 2), (2, 2)), 'constant')
    patches = np.zeros((m*n, 25))
    for i in range(m):
        for j in range(n):
            patches[i*n+j] = X[i:i+5,j:j+5].reshape(25)
    return patches

row, col = trainData.shape
trainNew = np.zeros((row, col, 25))
for i in range(trainData.shape[0]):
    temp = get_patches(trainData[i, :].reshape(28, 28))
    trainNew[i, :, :] = temp[np.newaxis, :, :]

row, col = test.shape
testNew = np.zeros((row, col, 25))
for i in range(test.shape[0]):
    temp = get_patches(test[i, :].reshape(28, 28))
    testNew[i, :, :] = temp[np.newaxis, :, :]

output = np.zeros((row, col))
for i in range(trainData.shape[1]):
    svr_rbf = SVR(kernel='rbf', C=1e4, gamma=0.0005)
    svr_rbf.fit(trainNew[:, i, :], target[:, i])
    pixel1 = svr_rbf.predict(np.matrix(testNew[:, i, :]))
    output[:, i] = pixel1*255.0


f = open('outPut', 'w')
f.write("Id,Val\n")
for i in range(0, test.shape[0]):
    for j in range(784):
        f.write(str(i))
        f.write("_")
        f.write(str(j))
        f.write(",")
        f.write(str(int(output[i, j])))
        f.write("\n")
f.close()
```

5) **Question 5.**

(a)

$$
\begin{aligned}
&\frac{1}{N}\tilde{X}\tilde{X}^\top = I \\
&\frac{1}{N}DX(DX)^\top = I \\
&\frac{1}{N}DXX^\top D^\top = I \\
&XX^\top = Q\Lambda Q^{-1} = Q\Lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}}Q^{-1} \\
&\frac{1}{N}D\left(Q\Lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}}Q^{-1}\right)D^\top = I \\
&\frac{1}{N}DQ\Lambda^{\frac{1}{2}}\left(DQ^{-1}\Lambda^{\frac{1}{2}}\right)^\top = I \\
&D = \sqrt{N}(\Lambda^{\frac{1}{2}})^{-1}Q^{-1} \\
&\frac{1}{N}DXX^\top D^\top = \frac{1}{N}\left(\sqrt{N}(\Lambda^{\frac{1}{2}})^{-1}Q^{-1}\right)\left(Q\Lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}}Q^{-1}\right)\left(\sqrt{N}(\Lambda^{\frac{1}{2}})^{-1}Q^{-1}\right)^\top = I
\end{aligned}
$$

(b)

Figure 3: $J(y)$ vs. θ

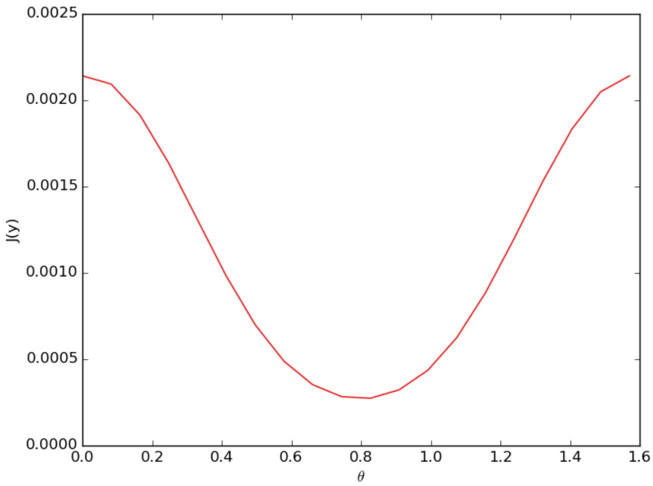
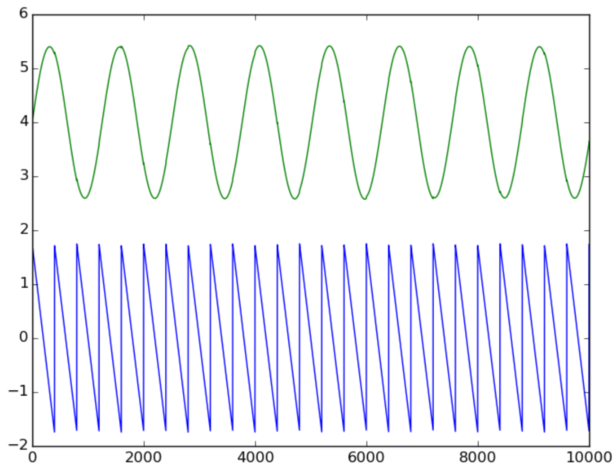


Figure 4: Recovered Y



```python
from __future__ import division
import numpy as np
from numpy import linalg as LA
from math import log, pi
from matplotlib import pyplot as plt

# Generate the data according to the specification in the homework description
N = 10000

# Here's an estimate of gamma for you
G = lambda x: np.log(np.cosh(x))
gamma = np.mean(G(np.random.randn(10**6)))

s1 = np.sin((np.arange(N)+1)/200)
s2 = np.mod((np.arange(N)+1)/200, 2) - 1
S = np.concatenate((s1.reshape((1,N)), s2.reshape((1,N))), 0)

A = np.array([[1,2],[-2,1]])

X = A.dot(S)
# TODO: Implement ICA using a 2x2 rotation matrix on a whitened version of X
lamdaInit, eVectors = LA.eig(np.dot(X, X.transpose()))
lamda = np.zeros((2, 2))
np.fill_diagonal(lamda, lamdaInit)
D = np.sqrt(N) * np.dot(LA.inv(np.sqrt(lamda)), LA.inv(eVectors))
whitenedX = np.dot(D, X)

jays = []
row1 = []
row2 = []
Theta = np.linspace(0.0, pi/2, num=20)
minTheta = 0.0
for theta in Theta:
    wMat = np.zeros((2, 2))
    wMat[0, 0] = wMat[1, 1] = np.cos(theta)
    wMat[0, 1] = -1 * np.sin(theta)
    wMat[1, 0] = np.sin(theta)
    yMat = np.dot(wMat, whitenedX)
    meanY1 = np.mean(G(yMat[0, :]))
    meanY2 = np.mean(G(yMat[1, :]))
    row1.append(meanY1)
    row2.append(meanY2)

    diff1 = (meanY1 - gamma)**2
    diff2 = (meanY2 - gamma)**2
    jay = diff1 + diff2
    jays.append(jay)

minIndex = np.argmax(jays)
minTheta = Theta[minIndex]
wMat[0, 0] = wMat[1, 1] = np.cos(minTheta)
wMat[0, 1] = -1 * np.sin(minTheta)
wMat[1, 0] = np.sin(minTheta)
yMat = np.dot(wMat, whitenedX)

plt.plot(Theta, jays, 'r')
plt.ylabel('J(y)')
plt.xlabel('$\\theta$')
plt.savefig('plot-1.png')
plt.clf()

plt.plot(yMat[0, :], 'b')
plt.plot(yMat[1, :]+4, 'g')
plt.savefig('plot-2.png')
```