EECS 545 – Machine Learning - Homework #4

Mina Jafari
group member: Joshua Kammeraad

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

$$
\begin{aligned}
I(X,Y) &= -\int\int p(x,y)\ln\frac{p(x)p(y)}{p(x,y)}dxdy \\
&= -\int\int p(x,y)\ln(p(x)p(y))dxdy + \int\int p(x,y)\ln p(x,y)dxdy \\
&= -\int\int p(x,y)\ln(p(x)p(y))dxdy + \int\int p(x,y)\ln(p(x|y)p(y))dxdy \\
&= -\int\int p(x,y)\ln p(x)dxdy - \int\int p(x,y)\ln p(y)dxdy \\
&+ \int\int p(x,y)\ln p(x|y)dxdy + \int\int p(x,y)\ln p(y)dxdy \\
&= H(X) - H(X|Y) \\
I(X,Y) &= -\int\int p(x,y)\ln\frac{p(x)p(y)}{p(x,y)}dxdy \\
&= -\int\int p(x,y)\ln(p(x)p(y))dxdy + \int\int p(x,y)\ln p(x,y)dxdy \\
&= -\int\int p(x,y)\ln(p(x)p(y))dxdy + \int\int p(x,y)\ln(p(y|x)p(x))dxdy \\
&= -\int\int p(x,y)\ln p(x)dxdy - \int\int p(x,y)\ln p(y)dxdy \\
&+ \int\int p(x,y)\ln p(y|x)dxdy + \int\int p(x,y)\ln p(x)dxdy \\
&= H(Y) - H(Y|X)
\end{aligned}
$$

(b)

$$
\begin{aligned}
I(X,Y) &= H(X) - H(X|Y) = H(X) - H(f(Y)|Y) = H(X) \\
I(X,Y) &= H(Y) - H(Y|X) = H(Y) - H(f(X)|X) = H(Y)
\end{aligned}
$$

(c)

(d)

$$H[x] = -\int p(x)\ln p(x)dx$$
$$\int_{-\infty}^{\infty} p(x)dx = 1,\ \int_{-\infty}^{\infty} xp(x)dx = \mu,\ \int_{-\infty}^{\infty} (x-\mu)^2 p(x)dx = \sigma^2$$
$$\mathcal{L}(x,\lambda_1,\lambda_2,\lambda_3) = -\int_{-\infty}^{\infty} p(x)\ln p(x)dx + \lambda_1\left(\int_{-\infty}^{\infty} p(x)dx - 1\right) + \lambda_2\left(\int_{-\infty}^{\infty} xp(x)dx - \mu\right)$$
$$+ \lambda_3\left(\int_{-\infty}^{\infty} (x-\mu)^2 p(x)dx - \sigma^2\right)$$
$$\frac{\partial}{\partial p(x)}\mathcal{L}(x,\lambda_1,\lambda_2,\lambda_3) = -\ln p(x) - 1 + \lambda_1 + \lambda_2 x + \lambda_3 (x-\mu)^2 = 0$$
$$\implies p(x) = \exp\left(-1 + \lambda_1 + \lambda_2 x + \lambda_3 (x-\mu)^2\right)$$

using Mathematica to solve the integrals:

$$\implies \lambda_1 = 1 - \frac{1}{4}\ln(4\pi^2\sigma^4), \lambda_2 = 0, \lambda_3 = \frac{-1}{2\sigma^2}$$
$$\implies p(x) = \exp\left(-1 + 1 - 1/4\ln(4\pi^2\sigma^4) + 0 + \frac{-(x-\mu)^2}{2\sigma^2}\right)$$
$$= \frac{1}{(2\pi\sigma^2)^{1/2}}\exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$
$$\frac{\partial^2 H[x]}{\partial p(x)^2} = \frac{\partial^2(-p(x)\ln p(x))}{\partial p(x)^2} = \frac{-1}{p(x)},\ 0 < p(x) \le 1$$

$\implies H''[x]$is concave and Gaussian is the maximum entropy distribution $\implies H(q) \le H(p)$

2) **Question 2.**

(a)

$$
\begin{aligned}
\text{Dirichlet}(\mathbf{p}|\alpha) &= \frac{\Gamma(\sum_{k=1}^{m}\alpha_k)}{\prod_{k=1}^{m}\Gamma(\alpha_k)}\prod_{k=1}^{m}p_k^{\alpha_k-1} \\
B &= \frac{\Gamma(\sum_{k=1}^{m}\alpha_k)}{\prod_{k=1}^{m}\Gamma(\alpha_k)} \\
\text{Dirichlet}(\mathbf{p}|\alpha) &= \exp\big[\log B + \sum_{k=1}^{m}(\alpha_k-1)\log p_k\big] \\
&= \exp\big[\log B + \eta(\alpha)^\top T(\mathbf{p})\big] \\
&= \exp\big[\eta(\alpha)^\top T(\mathbf{p}) - A(\alpha)\big] \\
&\eta(\alpha) = (\alpha_k - 1) \\
&T(\mathbf{p}) = \log p_k \\
&A(\alpha) = \log\frac{1}{B}
\end{aligned}
$$

(b)

$$
\begin{aligned}
F(\alpha) &= \log P(\mathcal{D}|\alpha) \\
&= \log\prod_{j=1}^{N}p(p^j|\alpha) = \log\prod_{j=1}^{N}\frac{\Gamma(\sum_{k=1}^{m}\alpha_k)}{\prod_{k=1}^{m}\Gamma(\alpha_k)}\prod_{k=1}^{m}(p_k^j)^{\alpha_k-1} \\
&= \sum_{j=1}^{N}\left[\log\frac{\Gamma(\sum_{k=1}^{m}\alpha_k)}{\prod_{k=1}^{m}\Gamma(\alpha_k)} + \sum_{k=1}^{m}(p_k^j)^{\alpha_k-1}\right] \\
&= \sum_{j=1}^{N}\left[\log\Gamma(\sum_{k=1}^{m}\alpha_k) - \sum_{k=1}^{m}\log\Gamma(\alpha_k) + \sum_{k=1}^{m}(\alpha_k-1)\log p_k^j\right] \\
&= N\left[\log\Gamma(\sum_{k=1}^{m}\alpha_k) - \sum_{k=1}^{m}\log\Gamma(\alpha_k) + \frac{1}{N}\sum_{k=1}^{m}(\alpha_k-1)\sum_{j=1}^{N}\log p_k^j\right] \\
&= N\left[\log\Gamma(\sum_{k=1}^{m}\alpha_k) - \sum_{k=1}^{m}\log\Gamma(\alpha_k) + \sum_{k=1}^{m}(\alpha_k-1)\hat{t}_k\right]
\end{aligned}
$$

(c)

$$
\begin{aligned}
\frac{\partial F}{\partial\alpha_i} &= \frac{\partial}{\partial\alpha_i}N\left[\log\Gamma(\sum_{k=1}^{m}\alpha_k) - \sum_{k=1}^{m}\log\Gamma(\alpha_k) + \sum_{k=1}^{m}(\alpha_k-1)\hat{t}_k\right] \\
&= N\left[\frac{\Gamma'(\sum_{k=1}^{m}\alpha_k)}{\Gamma(\sum_{k=1}^{m}\alpha_k)} - \frac{\Gamma'(\alpha_i)}{\Gamma(\alpha_i)} + \hat{t}_k\right] \\
&= N\left[\Psi(\sum_{k=1}^{m}\alpha_k) - \Psi(\alpha_i) + \hat{t}_k\right]
\end{aligned}
$$

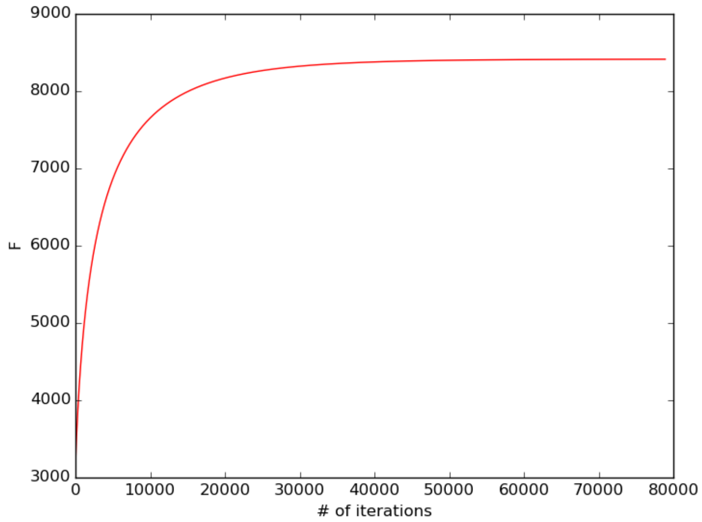
(d)

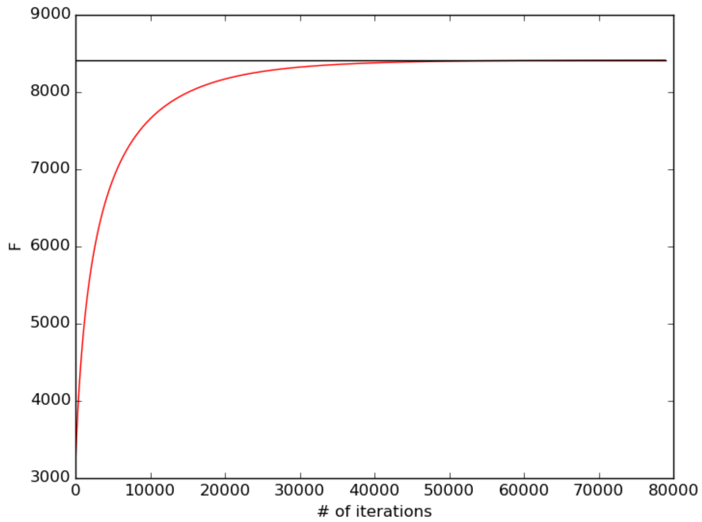
$$
\begin{aligned}
H \triangleq \nabla_\alpha^2 F(\alpha) &= \frac{\partial^2 F}{\partial \alpha_i \partial \alpha_j} = \frac{\partial}{\partial \alpha_j}\frac{\partial F}{\alpha_i} \\
&= \frac{\partial}{\partial \alpha_j} N\left[\Psi(\sum_{k=1}^{m}\alpha_k) - \Psi(\alpha_i) + \hat{t}_k\right] \\
&= N\left[\Psi'(\sum_{k=1}^{m}\alpha_k) - \delta_{ij}\Psi'(\alpha_i)\right] \\
&= Q + c11^T \\
q_{ij} &= -N\delta_{ij}\Psi'(\alpha_i),\ \delta_{ij} = \begin{cases} 0, & \text{if } i \neq j \\ 1, & \text{if } i = j \end{cases} \\
c &= N\Psi'(\sum_{k=1}^{m}\alpha_k)
\end{aligned}
$$

(e)

$$
\begin{aligned}
\alpha^{new} &= \alpha^{old} - \left[H_F(\alpha^{old})\right]^{-1} \cdot \nabla F(\alpha^{old}) \\
&= \alpha^{old} - \left[Q + c11^T\right]^{-1} \cdot \nabla F(\alpha^{old}) \\
&= \alpha^{old} - \left[Q^{-1} - \frac{Q^{-1}\cdot 1 \cdot 1^\top Q^{-1}}{\frac{1}{c} + 1^\top \cdot Q^{-1} \cdot 1}\right] \cdot \nabla F(\alpha^{old}) \\
\nabla F(\alpha) &= \begin{bmatrix} \frac{\partial F}{\partial \alpha_1} \\ \frac{\partial F}{\partial \alpha_2} \\ \vdots \\ \frac{\partial F}{\partial \alpha_m} \end{bmatrix}
\end{aligned}
$$

(f)





$$\alpha = [9.96873078, 4.9528229, 14.60503616, 19.4465898, 48.57156412]$$

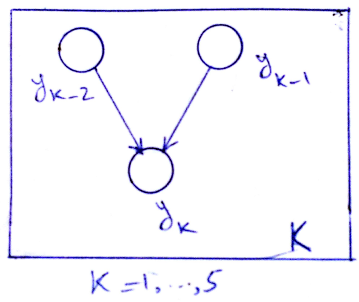
```
from __future__ import division
import numpy as np
from scipy.special import gammaln, polygamma
from math import log
from matplotlib import pyplot as plt
from numpy import linalg as LA
import copy

def computeF(alpha, P, N, m):
    sumAlpha = np.sum(alpha)
    B = 0.0
    D = 0.0
    for k in range(m):
        C = 0.0
        B += gammaln(alpha[k])
        for j in range(N):
            C += log(P[j][k])
        D += (alpha[k]-1) * (1.0/N)*C
    F = N * (gammaln(sumAlpha) - B + D)
    return F

def computeFPrime(alpha, P, N, m):
    Fprime = np.zeros((m))
    sumAlpha = np.sum(alpha)
    A = polygamma(0, sumAlpha)
    for k in range(m):
        C = 0.0
```

```
        for j in range(N):
            C += log(P[j][k])
        Fprime[k] = N * (A - polygamma(0, alpha[k]) + (1.0/N)*C)
    return Fprime

def computeHessian(alpha, P, N, m):
    sumAlpha = np.sum(alpha)
    tempHessians = np.zeros((m))
    for i in range(m):
        tempHessians[i] = -1 * N * (polygamma(1, alpha[i]))
    c = N * polygamma(1, sumAlpha)
    Q = np.diag(tempHessians)
    return (Q, c)


def main():
    N = 1000
    m = 5
    alpha_t = np.array([10, 5, 15, 20, 50])
    P = np.random.dirichlet(alpha_t, N)
    alpha_new = np.array([1, 1, 1, 1, 1])
    F = computeF(alpha_new, P, N, m)
    n = 0
    J = np.empty([1])
    ones = np.ones((m))
    onesT = np.transpose(ones)
    truePar = computeF(alpha_t, P, N, m)

    while True:
        FOld = F
        alpha_old = copy.deepcopy(alpha_new)
        Fprime = computeFPrime(alpha_old, P, N, m)
        Q, c = computeHessian(alpha_old, P, N, m)
        Qinv = LA.inv(Q)
        numerator = np.dot( np.dot(Qinv, onesT), np.dot(ones, Qinv)   )
        denom = (1.0/c) + np.dot(ones, np.dot(Qinv, onesT))
        middleTerm = np.subtract(Qinv, np.divide(numerator, denom))
        alpha_new = np.subtract(alpha_old, 0.001*np.dot(middleTerm, Fprime))
        F = computeF(alpha_new, P, N, m)

        if (abs(F - FOld) < 0.0001):
            print "***Alpha: ", alpha_new
            break

        n += 1
        J = np.append(J, F)

    J = np.delete(J, 0)
    x = range(1, n+1)
    plt.plot(x, J, 'r')
    plt.ylabel('F')
    plt.xlabel('# of iterations')
    plt.savefig('plot-1.png')

    plt.plot((1, n+1), (truePar, truePar), 'k-')
    plt.savefig('plot-2.png')

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

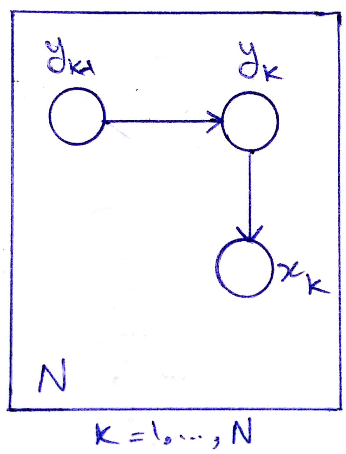
3) **Question 3.**

(a)

(i)



(ii)

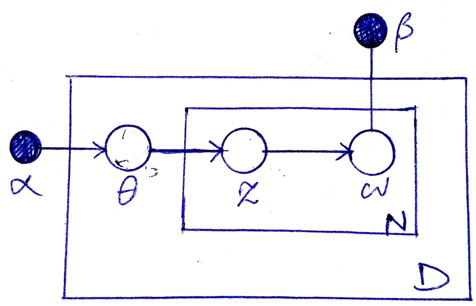


(b)

$$P(\gamma, \theta, \phi, Z) = \prod_{m=1}^{M} P(\gamma_m) P(\theta_m|\gamma_m) \prod_{n=1}^{N} P(\phi_{mn}) P(Z_{mn}|\phi_{mn})$$

$$P(Z, W) = \prod_{m=1}^{M} P(z_m) \prod_{n=1}^{N} P(w_{nm}|z_m)$$

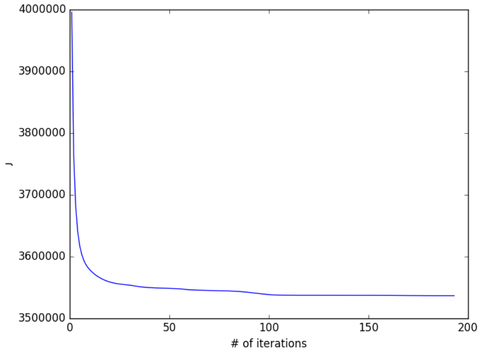
(c)



4) **Question 4.**

(a)

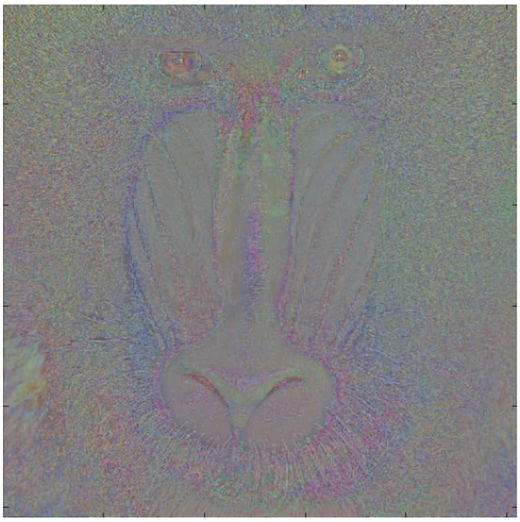
(i)



(ii)

The boundaries where there is a change in colors are not well preserved. Each individual color region (cluster) is preserved.

(iii)



(iv)

compression ratio $= \frac{\log_2 64}{24\times2\times2} = \frac{1}{16}$

(v)

Relative mean absolute error = 0.05011

(vi)

```
from __future__ import division
from scipy.ndimage import imread
import numpy as np
from matplotlib import pyplot as plt
from numpy import linalg as LA
import copy

# Load the mandrill image as an NxNx3 array. Values range from 0.0 to 255.0.
mandrill = imread('mandrill.png', mode='RGB').astype(float)
N = int(mandrill.shape[0])

M = 2
k = 64

# Store each MxM block of the image as a row vector of X
X = np.zeros((N**2//M**2, 3*M**2))
```

```python
for i in range(N//M):
    for j in range(N//M):
        X[i*N//M+j,:] = mandrill[i*M:(i+1)*M,j*M:(j+1)*M,:].reshape(3*M**2)

def clusterPoints(X, mu, k, M):
    Z = np.zeros((X.shape[0]), dtype=int)
    for i in range(X.shape[0]):
        A = np.zeros((k))
        for j in range(k):
            A[j] = LA.norm(np.subtract(X[i,:], mu[j,:]))
        Z[i] = np.argmin(A)
    return Z

def updateMu(Z, mu, k, M):
    counter = np.zeros((k), dtype=int)
    sumX = np.zeros((k, 3*M**2))
    for j in range(k):
        for i in range(X.shape[0]):
            if Z[i] == j:
                counter[j] += 1
                sumX[j,:] += X[i,:]
        if counter[j] != 0:
            mu[j,:] = sumX[j,:] / counter[j]
    return mu

oldmu = np.zeros((k, 3*M**2))
mu = np.zeros((k, 3*M**2))
mu = X[np.random.choice(X.shape[0], k, replace=False)]

J = np.empty([1])
n = 0
while True:
    if np.allclose(mu, oldmu, rtol=1e-3, atol=1e-8):
        break
    else:
        n += 1
        oldmu = copy.deepcopy(mu)
        Z = clusterPoints(X, mu, k, M)
        mu = updateMu(Z, mu, k, M)
        # draw the plot
        temper = 0
        for i in range(0, X.shape[0]):
            for j in range(0, k):
                if Z[i] == j:
                    temper += LA.norm(np.subtract(X[i,:], mu[j,:]))
        J = np.append(J, temper)

J = np.delete(J, 0)
x = range(1, n+1)
plt.plot(x, J, 'b')
plt.ylabel('J')
plt.xlabel('# of iterations')
plt.savefig('foo.png')


# reconstructing the image
temp = np.zeros((X.shape[0], 3*M**2))
for i in range(X.shape[0]):
    temp[i,:] = mu[Z[i],:]

mandrill_cons = np.zeros((N, N, 3))
for i in range(N//M):
    for j in range(N//M):
        mandrill_cons[i*M:(i+1)*M, j*M:(j+1)*M, 0:3] = temp[i*N//M+j,:].reshape(2,2,3)

plt.imshow(mandrill_cons/255)
plt.savefig('foo-2.png')

plt.imshow((mandrill - mandrill_cons + 128)/255)
plt.savefig('foo-3.png')


mae = np.sum(np.absolute(mandrill - mandrill_cons)) / (255 * 3*N**2)
print mae, "mae"
```

(b)

bpp $= \frac{\log_2 k}{M \cdot M}$

compression ratio $= \frac{\frac{\log_2 k}{M \cdot M}}{24} = \frac{\log_2 k}{24 M \cdot M}$

5) **Question 5.**

(a)

$$\begin{aligned}
\log f(\underline{y}|\underline{x};\theta) &= \log \prod_{n=1}^{N} f(y_n|x_n;\theta) = \log \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \phi(y; \mathbf{w}_k^T \mathbf{x} + b_k, \sigma_k^2) \\
&= \sum_{n=1}^{N} \log \sum_{k=1}^{K} \pi_k \phi(y; \mathbf{w}_k^T \mathbf{x} + b_k, \sigma_k^2)
\end{aligned}$$

(b)

$$\begin{aligned}
\log f(y_n, z_n|\mathbf{x}_n;\theta) &= \log \prod_{k=1}^{K} \pi_k \phi(y; \mathbf{w}_k^T \mathbf{x} + b_k, \sigma_k^2)^{\mathbb{I}[z_n=k]} \\
&= \sum_{k=1}^{K} \mathbb{I}[z_n = k] \log(\pi_k \phi(y; \mathbf{w}_k^T \mathbf{x} + b_k, \sigma_k^2)) \\
\log f(\underline{y}, \underline{z}|\underline{\mathbf{x}};\theta) &= \sum_{n=1}^{N} \log f(y_n, z_n|\mathbf{x}_n;\theta) \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbb{I}[z_n = k] \log(\pi_k \phi(y; \mathbf{w}_k^T \mathbf{x} + b_k, \sigma_k^2)) \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} \Delta_{nk} \log(\pi_k \phi(y; \mathbf{w}_k^T \mathbf{x} + b_k, \sigma_k^2))
\end{aligned}$$

(c)

$$\begin{aligned}
P(z_n = k|y_n, \theta) &= \frac{P(z_n = k|\theta)P(y_n|z_n = k, \theta)}{p(y_N|\theta)} = \frac{\pi_k \phi(y_n; \mathbf{w}_k^T \mathbf{x}_n + b_k, \sigma_k^2)}{\sum_{k'=1}^{K} \pi_k' \phi(y_n; \mathbf{w}_k^T \mathbf{x}_n + b_{k'}, \sigma_{k'}^2)} \\
Q(\theta, \theta^{\text{old}}) &= \mathbb{E}_{\underline{z}} \left[\log f(\underline{y}, \underline{z}|\underline{\mathbf{x}};\theta)|\underline{y}, \underline{\mathbf{x}};\theta^{\text{old}}\right] \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbb{E}_{\underline{z}} \left[\Delta_{nk} \log(\pi_k \phi(y_n; \mathbf{w}_k^T \mathbf{x}_n + b_k, \sigma_k^2))\right] \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbb{E}_{\underline{z}} \left[\Delta_{nk}\right] \log(\pi_k \phi(y_n; \mathbf{w}_k^T \mathbf{x}_n + b_k, \sigma_k^2)) \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} P(z_n = k|y_n, \theta) \log(\pi_k \phi(y_n; \mathbf{w}_k^T \mathbf{x}_n + b_k, \sigma_k^2)) \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \log \pi_k + \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \log(\phi(y_n; \mathbf{w}_k^T \mathbf{x}_n + b_k, \sigma_k^2))
\end{aligned}$$

(d)

$$\Lambda(\pi, \lambda) = Q(\theta, \theta^{\text{old}}) + \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right)$$

$$\frac{\partial \Lambda}{\partial \pi_j} = \sum_{n=1}^{N} r_{nj} \frac{1}{\pi_j} + \lambda = 0$$

$$\sum_{n=1}^{N} r_{nk} \frac{1}{\pi_k} + \lambda = 0 \implies \sum_{n=1}^{N} r_{nk} = -\lambda \pi_k \implies \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} = -\lambda \sum_{k=1}^{K} \pi_k \implies \lambda = -N$$

$$\sum_{n=1}^{N} r_{nk} \frac{1}{\pi_k} = N \implies \pi_k = \frac{1}{N} \sum_{n=1}^{N} r_{nk}$$

$$Q(\theta, \theta^{\text{old}}) = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \log \pi_k + \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \log(\phi(y_n; \mathbf{w}_k^T \mathbf{x}_n + b_k, \sigma_k^2))$$

$$= \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \log \pi_k + \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \log \left(\frac{1}{(\sigma_k \sqrt{2\pi})^d} e^{-1/2(\frac{||\mathbf{w}_k^T \mathbf{x}_n + b_k - y_n||}{\sigma_k})^2} \right)$$

$$= \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \log \pi_k + \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \log \left(\frac{1}{(\sigma_k \sqrt{2\pi})^d} \right) +$$

$$\sum_{n=1}^{N} \sum_{k=1}^{K} \frac{-r_{nk}}{2\sigma_k^2} (\mathbf{w}_k^T \mathbf{x}_n + b_k - y_n)^2$$

$\tilde{\mathbf{w}}_k = \{\mathbf{w}_k, b_k\}, \tilde{\mathbf{x}_n} = \{\mathbf{x}_n, 1\}$

we only care about the term dependent on $\mathbf{w}, b_k, \sigma^2 \implies$

$$g(\theta, \theta^{old}) = \sum_{n=1}^{N} \sum_{k=1}^{K} \frac{-r_{nk}}{2\sigma_k^2} (\mathbf{w}_k^T \mathbf{x}_n + b_k - y_n)^2$$

$= (\tilde{X}\tilde{\mathbf{w}}_k - y)^T R_k (\tilde{X}\tilde{\mathbf{w}}_k - y), R_k =$ a diagonal N by N matrix where $r_{nn} = \dfrac{-r_k}{2\sigma_k^2}$

$$\frac{\partial g(\theta, \theta^{old})}{\partial \tilde{\mathbf{w}}_k} = 0 \implies$$

$$\tilde{\mathbf{w}}_k^* = (\tilde{X}^\top R_k \tilde{X})^{-1} \tilde{X}^\top R_k y = \{\mathbf{w}_k^*, b_k^*\}$$

$$\frac{\partial Q(\theta, \theta^{\text{old}})}{\partial \sigma_k^2} = \frac{\partial}{\partial \sigma_K^2} \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \log \pi_k + \frac{\partial}{\partial \sigma_k^2} \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \log \left(\frac{1}{\sqrt{(2\pi)^d \sigma_k^2}} \right) +$$

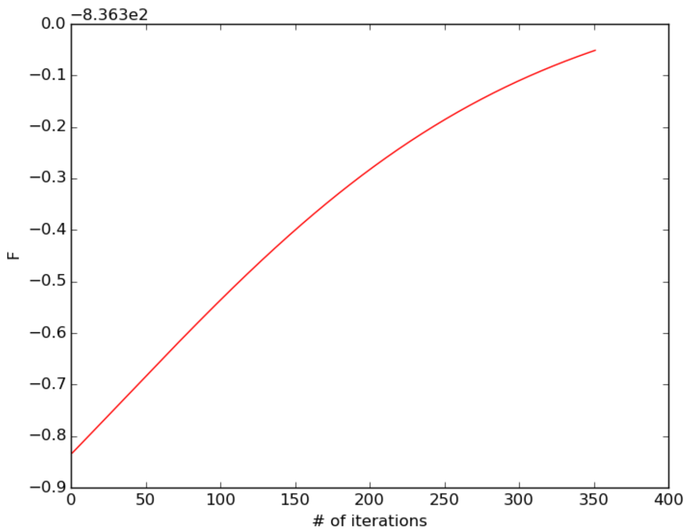
$$\frac{\partial}{\partial \sigma_k^2} \sum_{n=1}^{N} \sum_{k=1}^{K} \frac{-r_{nk}}{2\sigma_k^2} (\tilde{\mathbf{w}}_k \tilde{x}_n - y_n)^2$$

$$= \sum_{n=1}^{N} \frac{-r_{nk}}{2\sigma_k^2} + \sum_{n=1}^{N} \frac{r_{nk}}{2\sigma_k^4} (\tilde{\mathbf{w}}_k \tilde{x}_n - y_n)^2 = 0$$

$$\implies \sigma_k^2 = \frac{(\tilde{X}\tilde{\mathbf{w}}_k - y)^T R_k (\tilde{X}\tilde{\mathbf{w}}_k - y)}{r_k}$$

(e)

I have some bugs in my code and I ran out while dedugging :(



w: [0. 0.] b: [0. 0.] pi: [0.09001366, 0.90998634] sigma: [13.33434172, 13.33434172]

```
from __future__ import division
import numpy as np
from matplotlib import pyplot as plt
from math import exp, pi, log
from numpy.linalg import inv
from copy import deepcopy



# Generate the data according to the specification in the homework description

N = 500
x = np.random.rand(N)
print x.shape

pi0 = np.array([0.7, 0.3])
w0 = np.array([-2, 1])
b0 = np.array([0.5, -0.5])
sigma0 = np.array([.4, .3])

y1 = np.zeros_like(x)
for i in range(N):
    k = 0 if np.random.rand() < pi0[0] else 1
    y1[i] = w0[k]*x[i] + b0[k] + np.random.randn()*sigma0[k]

y = np.zeros((N, 1))
for j in range(N):
    y[j] = y1[j]


# TODO: Implement the EM algorithm for Mixed Linear Regression based on observed
# x and y values.

def computeF(N, K, piVector, sigma, w, x, b, y):
    outerSum = 0.0
    for n in range(N):
        pSum = 0.0
        for k in range(K):
            product = float(piVector[k]) * (1/float(sigma[k])*np.sqrt(2*pi)) * \
                    exp( -1*(float(w[k])*x[n]+b[k]-y[n])**2/(2*sigma[k]**2) )
            #print product
            pSum += product
        #print pSum
        outerSum += log(pSum)
    return outerSum

def computePhi(sigma, w, x, b, y):
    product = (1/sigma*np.sqrt(2*pi)) * exp( -1*(w*x+b-y)**2/(2*sigma**2) )
    return product
```

```python
def computeR(piVector, sigma, w, x, b, y, N):
    r0 = 0.0
    r1 = 0.0
    for n in range(N):
        r0 += piVector[0] * computePhi(sigma[0], w[0], x[n], b[0], y[0]) / \
                (piVector[0] * computePhi(sigma[0], w[0], x[n], b[0], y[0]) \
                + piVector[1] * computePhi(sigma[1], w[1], x[n], b[1], y[1]))
        r1 += piVector[1] * computePhi(sigma[1], w[1], x[n], b[1], y[1]) / \
                (piVector[0] * computePhi(sigma[0], w[0], x[n], b[0], y[0]) \
                + piVector[1] * computePhi(sigma[1], w[1], x[n], b[1], y[1]))

    return (r0, r1)

def computeWTilde(x, y, r0, r1, N, sigma):
    temp0 = np.zeros((N, N))
    np.fill_diagonal(temp0, -1*r0/(2*sigma[0]**2))
    temp1 = np.zeros((N, N))
    np.fill_diagonal(temp1, -1*r1/(2*sigma[1]**2))

    ones = np.ones((N))
    xTilde = np.vstack((x,ones))
    #print xTilde
    #print 'xtlide', xTilde.shape
    #print 'temp0', temp0.shape
    #print np.dot(temp0, xTilde).shape
    #print np.transpose(xTilde).shape

    first0 = inv(np.dot( xTilde, np.dot(temp0, np.transpose(xTilde))))
    second0 = np.dot( xTilde, np.dot(temp0, y) )
    wb0 = np.zeros((2))
    wb0 = np.dot(first0, second0)

    first1 = inv(np.dot( xTilde, np.dot(temp1, np.transpose(xTilde))))
    second1 = np.dot( xTilde, np.dot(temp1, y) )
    wb1 = np.zeros((2))
    wb1 = np.dot(first1, second1)

    return (wb0, wb1)


def computeSigmaSqr(x, N, y, w0, b0, w1, b1, r0, r1):
    ones = np.ones((N))
    xTilde = np.vstack((x, ones))
    #wTilde0 = np.array ([[w0], [b0]])
    #wTilde1 = np.array ([[w1], [b1]])
    wTilde0 = np.array ([w0, b0])
    wTilde1 = np.array ([w1, b1])

    R0 = np.zeros((N,N))
    np.fill_diagonal(R0, r0)
    R1 = np.zeros((N,N))
    np.fill_diagonal(R1, r1)

    '''
    print "&&&&&&&&&&&&&&&&&&&&&&&&&&"
    print y.shape
    print np.transpose(xTilde).shape
    print wTilde0.shape
    print  (np.dot(np.transpose(xTilde), wTilde0)-y).shape
    print "&&&&&&&&&&&&&&&&&&&&&&&&&&"
    '''
    innerVar0  = np.dot(np.transpose(xTilde), wTilde0)-y
    innerVar1  = np.dot(np.transpose(xTilde), wTilde1)-y

    num0 = np.dot(np.transpose(innerVar0), np.dot (R0, innerVar0))
    num1 = np.dot(np.transpose(innerVar1), np.dot (R1, innerVar1))

    sigma0 = num0 / r0
    sigma1 = num1 / r1


    return (float(sigma0), float(sigma1))



piHat = np.array([0.5, 0.5])
```

```
wHat = np.array([1, -1])
bHat = np.array([0, 0])
sigmaHat = np.array([np.std(y), np.std(y)])
r0, r1 = computeR(piHat, sigmaHat, wHat, x, bHat, y, N)

F = computeF(N, 2, piHat, sigmaHat, wHat, x, bHat, y)
print "piCurrent: ", piHat
print "sigmaCurrent: ", sigmaHat
print "r0: ", r0
print "r1: ", r1
print "F: ", F
print "=================================="
n = 0
piCurrent = np.zeros(2)

w0Current = np.zeros(2)
w1Current = np.zeros(2)
sigmaOld = deepcopy(sigmaHat)
J = np.empty([1])

while True:
    FOld = F
    r0Old = r0
    r1Old = r1
    piCurrent[0] = r0Old/N
    piCurrent[1] = r1Old/N

    wb0Current, wb1Current = computeWTilde(x, y, r0, r1, N, sigmaOld)
    sigmaCurrent = np.sqrt ( computeSigmaSqr(x, N, y, wb0Current[0], wb0Current[1], wb1Current[0], wb1Curr

    sigmaOld = deepcopy(sigmaCurrent)


    F = computeF(N, 2, piCurrent, sigmaCurrent,np.array([wb0Current[0], wb1Current[0]]), x, np.array([wb1C
    r0, r1 = computeR(piCurrent, sigmaCurrent, np.array([wb0Current[0], wb1Current[0]]), x, np.array([wb1C



    print "piCurrent: ", piCurrent
    print "sigmaCurrent: ", sigmaCurrent
    print "wb0Current: ", wb0Current[0]
    print "wb1Current: ", wb1Current


    print "r0: ", r0
    print "r1: ", r1
    print "F: ", F
    print "FOld: ", FOld
    print "==================================\n"
    J = np.append(J, F)
    n += 1
    if abs(F - FOld) < 0.001:
        print "w: ", w0Current, " ", w1Current, "pi: ", piCurrent, "sigma: ", sigmaCurrent
        break

print J.shape

J = np.delete(J, 0)
kkkkk = range(1, n+1)
plt.plot(kkkkk, J, 'r')
plt.ylabel('F')
plt.xlabel('# of iterations')
plt.savefig('plot-1.png')

# Here's the data plotted
plt.scatter(x, y, c='r', marker='x')
#plt.show()
```