1 Problem 1

1.1 Part a

We are asked to show that:

$$I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \tag{1}$$

Since there is no state specified, I will consider the discrete case. With this being said, I will start with $H(X) - H(X|Y)$. From the definition, it is easy to see this is equivalent to:

$$-\sum_X P(x)\log(P(x)) - \sum_X \sum_Y P(x,y)\log\left(\frac{P(y)}{P(x,y)}\right) \tag{2}$$

At this point, we use the fact that $\sum\limits_Y P(x,y) = P(X)$ to simplify the above to

$$-\sum_X \sum_Y P(x,y)\log(P(x)) - \sum_X \sum_Y P(x)\log\left(\frac{P(y)}{P(x,y)}\right) \tag{3}$$

$$= -\sum_X \sum_Y P(x,y)\left(\log(P(x)) - \log P(y) + \log P(x,y)\right) \tag{4}$$

Using the properties of log, this is simplified to

$$\sum_X \sum_Y P(x,y)\log\left(\frac{(P(x)P(y)}{P(x,y)}\right) \tag{5}$$

The proof for $H(Y) - H(Y|X)$ follows the exact same pattern as the above, with the x and y variables switched. Therefore, it will not be shown.

1.2 Part b

We are asked to prove that if X and Y are related via a bijection (i.e., $X = f(Y)$ and $Y = f^{-1}(x)$), then $I(X,Y) = H(X) = H(Y)$. What this means is that, via the above problem, we must show $H(X|Y) = H(Y|X) = 0$. From the above, we know:

$$H(X|Y) = -\sum_X \sum_Y P(x,y)\log\left(\frac{P(y)}{P(x,y)}\right) \tag{6}$$

with the bijection, this is equivalent to

$$H(X|Y) = -\sum_X \sum_Y P(x,y)\log\left(\frac{P(y)}{P(f(y),y)}\right) \tag{7}$$

It is easy to see that, since $f(y)$ is just a function of y, that $P(f(y),y) = P(y)$ and therefore the above becomes

$$H(X|Y) = -\sum_X \sum_Y P(x,y)\log(1) = 0 \tag{8}$$

An analogous proof will hold for $H(Y|X)$, with $f(y)$ replaced with $f^{-1}(x)$ and the same logic holding since $P(x, f^{-1}(x)) = P(x)$.

1.3 Part c

For a given

$$\hat{p} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[x = x_i] \tag{9}$$

In this section we are asked to show the minimum Kullback-Leiber divergence

$$\min_{\theta} D_{KL}(\hat{p}||q) \tag{10}$$

is equivalent to the maximum likelihood estimate θ_{ML} given the data $\mathcal{D}$. From definition, D_{KL} is

$$D_{KL}(\hat{p}||q) = \sum_{x \in X} \hat{p}(x) log \frac{\hat{p}(x)}{q(x)} \tag{11}$$

which, with our current definitions, is

$$\sum_{i=1}^{N} \left(\mathbb{I}[x = x_i] log \left(\mathbb{I}[x = x_i]\right) - log(q(x))\right) \tag{12}$$

taking the minimum with respect to theta, we see we must find

$$\sum_{i=1}^{N} \left(\mathbb{I}[x = x_i] \frac{\partial log(q(x))}{\partial \theta}\right) = 0 \tag{13}$$

which equates to, which expanded,

$$\sum_{i=1}^{N} \left(\mathbb{I}[x = x_i] \frac{\partial log(q(x_i|\theta))}{\partial \theta}\right) = 0 \tag{14}$$

So now we must find the MLE of θ. The total conditional probability will be:

$$\prod_{i=1}^{N} q(x_i|\theta)^{\mathbb{I}[x=x_i]} \tag{15}$$

Taking the log of the above and setting it equal to zero, we get:

$$\sum_{i=1}^{N} \left(\mathbb{I}[x = x_i] \frac{\partial log(q(x_i|\theta))}{\partial \theta}\right) = 0 \tag{16}$$

which is the same as the above.

1.4 Part d

In this section, we are asked to prove that the Gaussian distribution has maximum entropy among all distributions of the same variance. To start, I will build off of the proof in the textbook outlined in section 1.6. In this proof, they rely on the three constraints

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{17}$$

$$\int_{-\infty}^{\infty} xp(x)dx = \mu \tag{18}$$

$$\int_{-\infty}^{\infty} (x-\mu)^2 p(x)dx = \sigma^2 \tag{19}$$

Which, using a Lagrange expression, can allow us to see, at maximum entropy.

$$p(x) = exp\{-1 + \lambda_1 + \lambda_2 x + \lambda_3 (x-\mu)^2\} \tag{20}$$

where $\lambda_1, \lambda_2, \lambda_3$ are Lagrange variables. Now, if we can solve for these Lagrange variables we can find the pdf of the maximum entropy exponential distribution To start, we consider the second constraint, i.e.,

$$\int_{-\infty}^{\infty} xp(x)dx = \mu \tag{21}$$

To start, we change variables, where $y = x - \mu + \frac{\lambda_2}{2\lambda_3}$ and therefore $dy = dx$. with this definition, we see

$$\begin{aligned} y^2 &= x^2 - 2x\mu + \mu^2 + x\frac{\lambda_2}{\lambda_3} - \mu\frac{\lambda_2}{\lambda_3} + \frac{\lambda_2^2}{4\lambda_3^2} && (22)\\ &= (x-\mu)^2 + x\frac{\lambda_2}{\lambda_3} - \mu\frac{\lambda_2}{\lambda_3} + \frac{\lambda_2^2}{4\lambda_3^2} && (23)\\ \Rightarrow p(y) &= exp\{1 + \lambda_1 + \lambda_3 y^2 + \mu\lambda_2 - \frac{\lambda_2^2}{4\lambda_3^2}\} && (24) \end{aligned}$$

and therefore, the integral is now

$$\int_{-\infty}^{\infty} p(y)\left(y + \mu - \frac{\lambda_2}{2\lambda_3}\right)dy = \mu \tag{25}$$

One important note is that, since $dy = dx$ and y is really just x shifted by some amount, the first integral, i.e.,

$$\int_{-\infty}^{\infty} p(y)dy \tag{26}$$

will still integrate out to 1. Therefore, in (25), we can expand $(y + \mu - \frac{\lambda_2}{2\lambda_3})$ and see

$$\int_{-\infty}^{\infty} p(y)\left(y + \mu - \frac{\lambda_2}{2\lambda_3}\right)dy = \mu \tag{27}$$

$$\int_{-\infty}^{\infty} p(y)ydy + \mu\int_{-\infty}^{\infty} p(y)dy - \frac{\lambda_2}{2\lambda_3}\int_{-\infty}^{\infty} p(y) = \mu \tag{28}$$

$$\int_{-\infty}^{\infty} p(y)ydy + \mu - \frac{\lambda_2}{2\lambda_3} = \mu \tag{29}$$

$$\int_{-\infty}^{\infty} p(y)ydy = \frac{\lambda_2}{2\lambda_3} \tag{30}$$

And now we can actually take the integral of $\int_{-\infty}^{\infty} p(y)ydy$.

$$\frac{\lambda_2}{2\lambda_3} = \frac{1}{2\lambda_3} exp\{1 + \lambda_1 + \lambda_3 y^2 + \mu\lambda_2 - \frac{\lambda_2^2}{4\lambda_3^2}\}\Big|_{-\infty}^{\infty} \tag{31}$$

$$\frac{\lambda_2}{2\lambda_3} = \frac{1}{2\lambda_3}\{exp\{\infty\} - exp\{\infty\}\} = 0 \tag{32}$$

and therefore, $\lambda_2 = 0$. Now, we can see $p(x) = exp\{-1 + \lambda_1 + \lambda_3(x-\mu)^2\}$ and the variance constraint can now be solved by integration by parts. Setting $z = (x-\mu)$ and, in the integration by parts, $u = z^2$ and $dv = exp\{-1 + \lambda_1 + \lambda_3(x-\mu)^2\}$, we get the equation:

$$\frac{1}{2\lambda_3} exp\{-1 + \lambda_1 + \lambda_3(x-\mu)^2\}\Big|_{-\infty}^{\infty} - \frac{1}{2\lambda_3}\int_{-\infty}^{\infty} exp\{-1 + \lambda_1 + \lambda_3(x-\mu)^2\}dx = \sigma^2 \tag{33}$$

$$\Rightarrow 0 - \frac{1}{2\lambda_3} = \sigma^2 \tag{34}$$

$$\Rightarrow \lambda_3 = -\frac{1}{2\sigma^2} \tag{35}$$

Now, we have $p(x) = exp\{-1 + \lambda_1 - \frac{(x-\mu)^2}{2\sigma^2}\}$ Now, we see the first constraint becomes

$$exp\{-1 + \lambda_1\}\int_{-\infty}^{\infty} exp\{-\frac{(x-\mu)^2}{2\sigma^2}\} = 1 \tag{36}$$

It is easy to see that the term inside the integral is the Gaussian distribution, which we know needs a normalization factor of $\frac{1}{\sqrt{2\pi\sigma^2}}$ in order to be equated to one. Therefore,

$$exp\{-1 + \lambda_1\} = \frac{1}{\sqrt{2\pi\sigma^2}} \tag{37}$$

$$\lambda_1 = 1 + log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) \tag{38}$$

and therefore, our overall pdf is

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left\{-\frac{\|x-\mu\|^2}{2\sigma^2}\right\} \tag{39}$$

and the maximum entropy exponential distribution is the Gaussian distribution and the proof is complete

2 Problem 2

We are given a Dirichlet function, i.e.,

$$\text{Dirichlet}(p|\alpha) = \frac{\Gamma\left(\sum_{k=1}^{m} \alpha_k\right)}{\prod_{k=1}^{m} \Gamma(\alpha_k)} \prod_{k=1}^{m} p_k^{\alpha_k - 1} \tag{40}$$

2.1 Part a

For this problem, we are asked to show that the Dirichlet probability density function is part of the exponential family and therefore can be decomposed as:

$$Dirichlet(p|\alpha) = exp[\eta(\alpha)^T T(p) - A(\alpha)] \tag{41}$$

with the Dirichlet defined as above, taking the exponential of the log of the Dirichlet allows us to simplify the above to:

$$exp\left[log\left(\Gamma\left(\sum_{k=1}^{m}\alpha_k\right)\right) - \sum_{k=1}^{m} log\left(\Gamma(\alpha_k)\right) + \sum_{k=1}^{m}(\alpha_k - 1)log(p^k)\right] \tag{42}$$

which can be reformulated as

$$exp[\eta(\alpha)^T T(p) - A(\alpha)] \tag{43}$$

where

$$\eta(\alpha) = \begin{bmatrix} \alpha_1 - 1 \\ \vdots \\ \alpha_N - 1 \end{bmatrix}, T(p) = \begin{bmatrix} log(p_1) \\ \vdots \\ log(p_N) \end{bmatrix} \tag{44}$$

$$\tag{45}$$

$$A(\alpha) = log\left(\Gamma\left(\sum_{k=1}^{m}\alpha_k\right)\right) - \sum_{k=1}^{m} log\left(\Gamma(\alpha_k)\right) \tag{46}$$

2.2 Part b

Int his section, we are given data $\mathcal{D}$ and asked to derive an expression for the Dirichlet log-likelihood function $F(\alpha) = logP(\mathcal{D}|\alpha)$ in terms of the observed sufficient statistics

$$\hat{t_k} = \frac{1}{N}\sum_{j=1}^{N} log\ p_k^{(j)} \tag{47}$$

For the data, it is easy to see the log likelihood function for one data sample would be very similar to the expression inside the exponential function above, i.e.,

$$log\left(\Gamma\left(\sum_{k=1}^{m}\alpha_k\right)\right) - \sum_{k=1}^{m} log\left(\Gamma(\alpha_k)\right) + \sum_{k=1}^{m}(\alpha_k - 1)log(p_k) \tag{48}$$

Therefore, for N many samples the above would be multiplied, i.e.,

$$log\left(\Gamma\left(\sum_{k=1}^{m}\alpha_k\right)\right)^N - \sum_{k=1}^{m} log\left(\Gamma(\alpha_k)\right)^N + \sum_{j=1}^{N}\sum_{k=1}^{m}(\alpha_k - 1)log(p_k^{(j)}) \tag{49}$$

from the properties of logarithm, this is simplified as

$$N\ log\left(\Gamma\left(\sum_{k=1}^{m}\alpha_k\right)\right) - N\ \sum_{k=1}^{m} log\left(\Gamma(\alpha_k)\right) + \sum_{j=1}^{N}\sum_{k=1}^{m}(\alpha_k - 1)log(p_k^{(j)}) \tag{50}$$

which can be reformulated as

$$F(\alpha) = N\left(\ log\left(\Gamma\left(\sum_{k=1}^{m}\alpha_k\right)\right) - \ \sum_{k=1}^{m} log\left(\Gamma(\alpha_k)\right) + \sum_{k=1}^{m}(\alpha_k - 1)\hat{t}_k\right) \tag{51}$$

2.3 Part c

For this section, we are asked to find the gradient with respect to α_k of the log-likelihood function (defined above). With $\Psi(t) = \Gamma'(t)/\Gamma(t)$, the above can be reformulated as:

$$\triangledown_{\alpha_k} F(\alpha) = N\left(\Psi\left(\sum_{k=1}^{m}\alpha_k\right) - \Psi(\alpha_k) + \hat{t}_k\right) \tag{52}$$

2.4 Part d

With the above defined, the Hessian can be defined, when $\alpha_i = \alpha_j$, as

$$\frac{\partial^2 F}{\partial \alpha_i^2} = N\left(\Psi'\left(\sum_{k=1}^{m}\alpha_k\right) - \Psi'(\alpha_i)\right) \tag{53}$$

and if $\alpha_i \neq \alpha_j$,

$$\frac{\partial^2 F}{\partial \alpha_i \alpha_j} = N\Psi'\left(\sum_{k=1}^{m}\alpha_k\right) \tag{54}$$

which means, we can define:

$$c = N\Psi'\left(\sum_{k=1}^{m}\alpha_k\right) \tag{55}$$

$$Q = N * \text{diag}(-\Psi'(\alpha_i)), \quad i = 1, ...m \tag{56}$$

2.5 Part e

The Sherman-Morrison Inversion Lemma states that, given an invertible matrix A and vectors u, v,

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u} \tag{57}$$

In our example, we define $A = Q$ and

$$u = v = \begin{bmatrix} \sqrt{N\Psi'\left(\sum_{k=1}^{m}\alpha_k\right)} \\ \vdots \\ \sqrt{N\Psi'\left(\sum_{k=1}^{m}\alpha_k\right)} \end{bmatrix} \tag{58}$$

and we can therefore define our Newton-Raphson step as

$$\alpha_{new} = \alpha_{old} - \left[A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u}\right] \begin{bmatrix} N\left(\Psi\left(\sum_{k=1}^{m}\alpha_k\right) - \Psi(\alpha_1) + \hat{t}_1\right) \\ \vdots \\ N\left(\Psi\left(\sum_{k=1}^{m}\alpha_k\right) - \Psi(\alpha_m) + \hat{t}_m\right) \end{bmatrix} \tag{59}$$

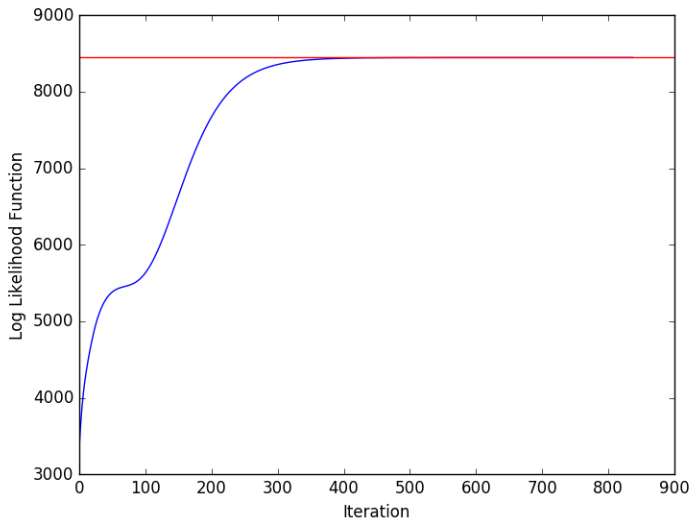
2.6 Part f

The code used is shown below:

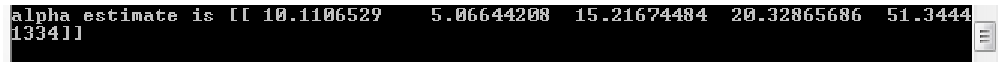
```
#Problem 2

from scipy.special import gammaln, polygamma, gamma
import numpy as np
import math
from matplotlib import pyplot as plt


N = 1000
alpha = np.array([10,5,15,20,50])
D =(np.random.dirichlet(alpha,size=(N,1)))

#print np.shape(D)
#print D[1][0]

ahat = np.array([1.0,1.0,1.0,1.0,1.0])
tk =  sum(np.log(D))/N
aTot = ahat
F       = 1
difference = 1
p=1
LogL=[]


while difference>10**(-5):
prev   = F
Sum_A  = np.sum(ahat)
Grad_F = N*(polygamma(0,Sum_A)-polygamma(0,ahat)+tk)
```

```
c      = N*polygamma(1,Sum_A)
q      = -N*(polygamma(1,ahat))
Qi     = np.zeros((5,5),float)
np.fill_diagonal(Qi,1/q)
U      = np.zeros((5,1),float)
u      = np.sqrt(c)
np.ndarray.fill(U,u)
V      = U
H      = Qi-(Qi*U*np.transpose(V)*Qi)/(1+
np.dot(np.dot(np.transpose(V),Qi),U))
Delta =  np.dot(H,np.transpose(Grad_F))
ahat  = ahat-0.1*np.transpose(Delta)
F     = N*(gammaln(Sum_A)-np.sum(gammaln(ahat))+np.sum((ahat-1)*tk))
current = F
difference = abs(current-prev)
print ahat
LogL.append(F)

LL  = N*(gammaln(np.sum(alpha))-np.sum(gammaln(alpha)))
print LL
print np.sum((alpha-1)*tk)
LL  = LL+ N*np.sum((alpha-1)*tk)

print "alpha estimate is", ahat

plt.plot(range(len(LogL)),LogL)
plt.axhline(y=LL, xmin=0, xmax=len(LogL), hold=None,color='r')
plt.xlabel('Iteration')
plt.ylabel('Log Likelihood Function')
plt.show()
```

With the resulting log-likelihood plot, with the constant red line shown being the log-likelihood of the true parameters.



This plot was done with the algorithm terminating when the difference in log-likelihood was less than 10^{-4}. The estimated parameters for the first plot are shown below: As a note, when the



```
alpha estimate is [[ 10.1106529    5.06644208  15.21674484  20.32865686  51.3444
1334]]
```

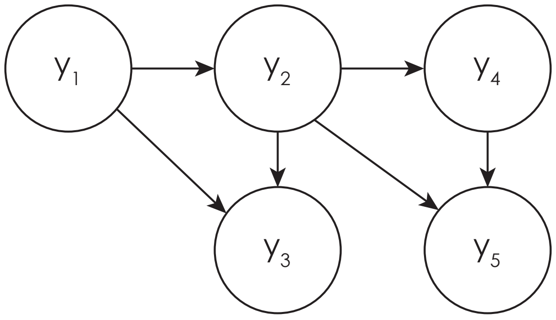
step in the Newton Raphson algorithm was 1, the algorithm wanted to go unstable. However, if I limited this step, the algorithm worked correctly and the parameters converged correctly. That is the reason for the coefficient in the code.

3 Problem 3

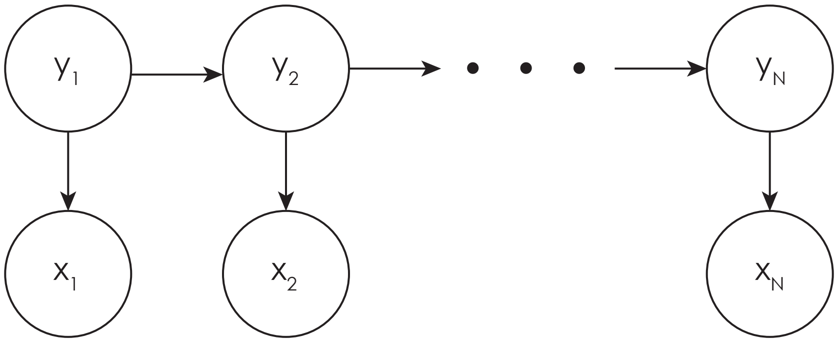
3.1 Part a

For this section, the plots will be shown without comment

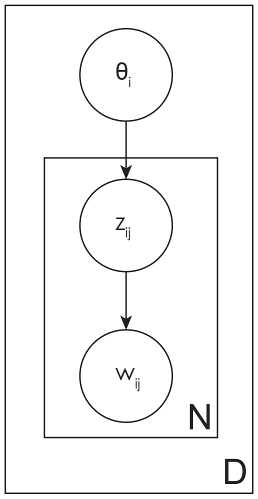
3.1.1 Part i



3.1.2 Part ii



3.2 Part b



3.3 Part c

3.3.1 Part i

My solution is:

$$\prod_{i=1}^{M}\left[P(\gamma_i)P(\theta_i|\gamma_i)\prod_{j=1}^{N}P(\phi_{ij})P(z_{ij}|\phi_{ij})\right] \tag{60}$$

3.3.2 Part ii

From the picture, one would think the solution would be:

$$\prod_{i=1}^{M}P(z_i)\prod_{j=1}^{N}P(w_{ij}|z_i) \tag{61}$$

Since w is now known, this equation doesn't quite make sense. From Bayes theorem, this can be reformulated as

$$\prod_{i=1}^{M}P(z_i)\prod_{j=1}^{N}\frac{P(z_i|w_{ij})P(w_{ij})}{P(z_i)} \tag{62}$$

$$\prod_{i=1}^{M}\prod_{j=1}^{N}P(z_i|w_{ij})P(w_{ij}) \tag{63}$$

Since w_{ij} is now known, we can reformulate it as the indicator function when $w_j = i$, i.e.

$$\prod_{i=1}^{M}\prod_{j=1}^{N}\mathbb{I}[w_j = i]P(z_i|w_{ij}) \tag{64}$$

4 Problem 4

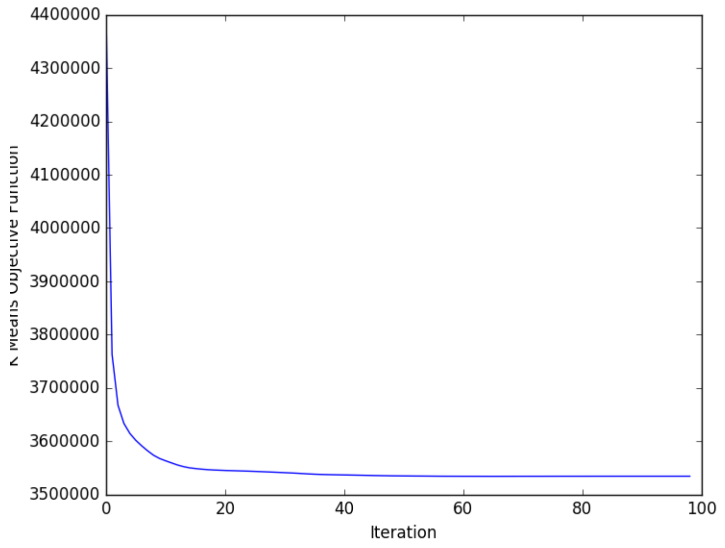
4.1 Part a

The code is shown below:

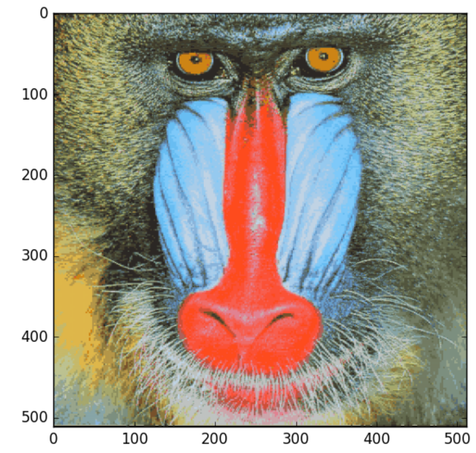
```
from __future__ import division
from scipy.ndimage import imread
import numpy as np
from matplotlib import pyplot as plt
import random

# 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))
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)

# TODO: Implement k-means and cluster the rows of X, then reconstruct the
# compressed image using the cluster center for each block, as specified in
# the homework description.


Mean    = np.random.choice(range(0,len(X)),size=(k,1),replace=False)
Initial = (X[Mean])


Diff = X[0]-Initial
Norm = np.sum(np.abs(Diff)**2,axis=-1)**(1./2)
Index  = np.where(Norm==Norm.min())
Num  = Index[0]




p=1

Count = np.zeros((1,k))
Total = np.zeros((k,3*M**2))
ErrorT=[]

while p<100:
    Count = 0.00001*np.ones((k,1))
    Error = 0
    Total = np.zeros((k,3*M**2))
    Final = np.zeros((len(X),3*M**2))
    for i in range(0,len(X)):
        xi    = np.array(X[i])
        Diff  = xi-Initial
        Norm  = np.sum(np.abs(Diff)**2,axis=-1)**(1./2)
        Index = np.where(Norm==Norm.min())
        Error = Error+Norm[Index]
        Num   = Index[0]
        Count[Num] = Count[Num]+1
        Total[Num] = Total[Num]+xi
        Final[i] = Initial[Num][0]

    Initial  = np.divide(Total,Count)
    ErrorT.append(Error[0])
    p=p+1


mandrill2 = np.zeros(np.shape(mandrill))
for i in range(N//M):
    for j in range(N//M):
        mandrill2[i*M:(i+1)*M,j*M:(j+1)*M,:]=Final[i*N//M+j,:].reshape(M,M,3)

(length,width,height) = np.shape(mandrill)

Abs_Error=0

for i in range(0,length):
    for j in range(0,width):
        for r in range(0,height):
            Abs_Error = abs(mandrill[i][j][r]-mandrill2[i][j][r])

Abs_Error = Abs_Error/(3*255*N**2)
print "The Relative Mean of the Absolute Error is",Abs_Error


# To show a color image using matplotlib, you have to restrict the color
# color intensity values to between 0.0 and 1.0. For example,

print np.shape(ErrorT)
print type(ErrorT)
print ErrorT[0:10]
print np.vstack(ErrorT[0:10])
plt.plot(range(len(ErrorT)),ErrorT)
plt.xlabel('Iteration')
plt.ylabel('K Means Objective Function')
plt.show()
plt.imshow(mandrill2/mandrill2.max())
plt.show()
plt.imshow(mandrill/255)
plt.show()
plt.imshow(mandrill/255-mandrill2/mandrill2.max()+(128,128,128))
plt.show()
```

With the following plots shown for the deliverables:

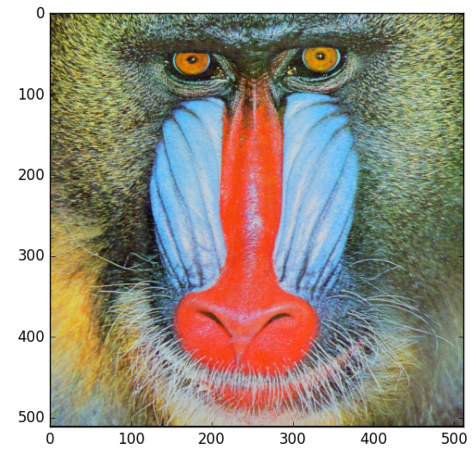
4.1.1 Plot of K-Means Objective Function



4.1.2 Two Resulting Picture and Comparison



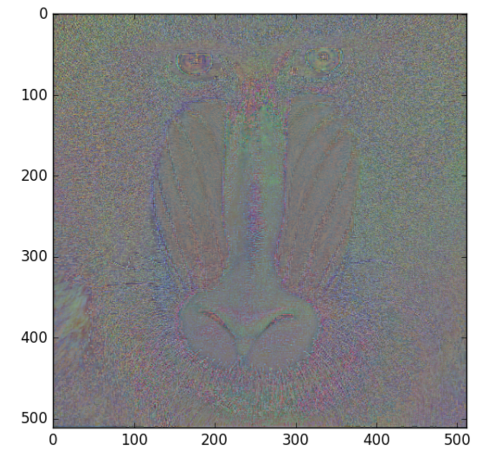
(a) Compressed Image



(b) Real Image

As shown, the two images are fairly similar. For the most part, the compressed image does a good job of recreating the image. However, there are areas where the K-Means algorithm does not recreate quite so effectively. For starters, the eyes of the compressed mandrill are much less sharp than those of the real image. Additionally, the blue and red of the nose/cheeks lose a bit of their sharpness. One thing that is truly noticeable are the whiskers on the bottom left of the image. While the real image has quite a few whiskers showing up, the compressed image has barely any. This becomes very evident when looking at the difference between the two images. The regions which are best preserved are the fur to the right and left of the eyes and the upper nose/eyes areas. These are the ones with the largest differences in color.

4.1.3 Picture of Difference of the Two Images

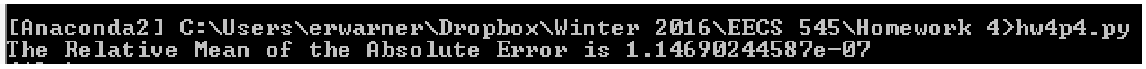


4.1.4 Compression ratio

In our original image, we have $N \times N \times 3 = 3N^2$ entries. In the vector of means, we need only $K \times 3M \times M = 3KM^2$ components. Additionally, as stated there are 24 bits per pixel. If we were to send this to a friend, we would also need to send them a matrix of indicators for which means the pixels are. To encode the 64 mean values, we would need $log_2 k$ bits per the size of the block grid. Since the block grids is N^2/M^2, that is the coefficient and the compression ratio will be

$$\frac{3M^2K + 3 * \frac{N^2}{M^2}}{3N^2} == \frac{24 * 2^2 * 64 + \frac{512^2}{2^2} log_2(64)}{24 * 512^2} = 0.0635 \tag{65}$$

4.1.5 Relative Mean Absolute Error



```
[Anaconda2] C:\Users\erwarner\Dropbox\Winter 2016\EECS 545\Homework 4>hw4p4.py
The Relative Mean of the Absolute Error is 1.14690244587e-07
```

4.2 Part b

As shown above, the compression ratio is:

$$\frac{3M^2K + 3 * \frac{N^2}{M^2}}{3N^2} log_2 k = \frac{24 * 2^2 * 64 + \frac{512^2}{2^2} log_2(64)}{24 * 512^2} = 0.0635 \tag{66}$$

Which means the bits per pixel would be:

$$24 * 0.0635 = 1.52 \;\; bits/pixel \tag{67}$$

and, rounding up, we would need around 2 bits/pixel.

5 Problem 5

5.1 Part a

In this section, we are asked to write down the formula for the log-likelihood of N many samples of x and y, where

$$f(y|x;\theta)\ \sum_{k=1}^{K} \pi_k \phi(y; w_k^T x + b_k, \sigma_k^2 \tag{68}$$

with N many samples and taking the log-likelihood, we get

$$\prod_{i=1}^{N} log \left(\sum_{k=1}^{K} \pi_k \phi(y_i; w_k^T x_i + b_k, \sigma_k^2 \right) \tag{69}$$

which, via the properties of logarithms, is equivalent to:

$$\sum_{i=1}^{N} log \left(\sum_{k=1}^{K} \pi_k \phi(y_i; w_k^T x_i + b_k, \sigma_k^2) \right) \tag{70}$$

5.2 Part b

We now introduce a hidden variable z which determines the mixture component that y is drawn from. Additionally, we can define a random variable $\Delta_{ik} = \mathbb{I}[z_i = k]$. Usually, this indicator function would multiply by the ϕ term. However, since it is the logarithm, if this indicator function was zero it would send the log to $-\infty$ and if it was 1 it would send it to zero. This is because Therefore, we put this indicator function as an exponent, i.e.,

$$\sum_{i=1}^{N} log \left(\sum_{k=1}^{K} \pi_k \phi(y_i; w_k^T x_i + b_k, \sigma_k^2)^{\Delta_{ik}} \right) \tag{71}$$

Now, since the indicator is in the exponent, we can see that summing $\pi_k \phi$ will only output one value, when $\Delta_{ik} = 1$. Therefore, taking the log of the sum will be equivalent to summing the logarithms with the indicator out in front. Therefore, just like previously only one logarithm term will show up per term in N. The others will still be sent to zero, and nothing changes. Therefore, this can be reformulated as

$$\sum_{i=1}^{N} \sum_{k=1}^{K} \Delta_{ik} log \left(\pi_k \phi(y_i; w_k^T x_i + b_k, \sigma_k^2) \right) \tag{72}$$

5.3 Part c

We are now asked to determine the E-step of our function, i.e., a formula for

$$Q(\theta, \theta^{old}) = \mathbb{E}_{\underline{z}}[log f(\underline{y}, \underline{z}|\underline{x}; \theta)|\underline{y}, \underline{x}, \theta^{old}] \tag{73}$$

by the definition of the log likelihood, this becomes

$$\sum_{i=1}^{N} \sum_{k=1}^{K} E_z \left[\Delta_{ik} log \left(\pi_k \phi(y_i; w_k^T x_i + b_k, \sigma_k^2) \right) \right] \tag{74}$$

Since the term inside the logarithm is not dependent on z, this is equivalent to

$$\sum_{i=1}^{N}\sum_{k=1}^{K} E_z\left[\Delta_{ik}\right] log\left(\pi_k \phi(y_i; w_k^T x_i + b_k, \sigma_k^2)\right) \tag{75}$$

Which can be separated to be

$$\sum_{i=1}^{N}\sum_{k=1}^{K} E_z\left[\Delta_{ik}\right] log\left(\pi_k\right) + \sum_{i=1}^{N}\sum_{k=1}^{K} E_z\left[\Delta_{ik}\right] log\left(\phi(y_i; w_k^T x_i + b_k, \sigma_k^2)\right) \tag{76}$$

To find $E_z[\Delta_{ik}] = p(z_n = k|x_n, \theta)$, we use Bayes rule, i.e.,

$$p(z_n = k|x_n, \theta) = \frac{P(z_n = k|\theta)p(x_n|z_n = k, \theta)}{p(x_N|\theta)} \tag{77}$$

$$= \frac{\pi_k \phi(y_i; w_k^T x_i + b_k, \sigma_k^2)}{\sum\limits_{j=1}^{K} \pi_j \phi(y_i; w_k^T x_i + b_k, \sigma_k^2)} \tag{78}$$

For simplicity, I will define $r_{nk} = \frac{\pi_k \phi(y_i; w_k^T x_i + b_k, \sigma_k^2)}{\sum\limits_{j=1}^{K} \pi_j \phi(y_i; w_k^T x_i + b_k, \sigma_k^2)}$. Therefore, the above summation becomes

$$\sum_{i=1}^{N}\sum_{k=1}^{K} r_{nk} log\left(\pi_k\right) + \sum_{i=1}^{N}\sum_{k=1}^{K} r_{nk} log\left(\phi(y_i; w_k^T x_i + b_k, \sigma_k^2)\right) \tag{79}$$

and this is the M-step

5.4 Part d

We must now determine the M-step, which means finding the optimal $\{\pi, w, b, \sigma\}$. To start, I will find the optimal π. To do so, I will use Lagrangian multipliers. Since we aim to minimize $Q(\theta, \theta^{old})$ and $\sum\limits_{k=1}^{K} \pi_k = 1$, the Lagrangian can be formulated as

$$L = \sum_{i=1}^{N}\sum_{k=1}^{K} r_{nk} log\left(\pi_k\right) + \sum_{i=1}^{N}\sum_{k=1}^{K} r_{nk} log\left(\phi(y_i; w_k^T x_i + b_k, \sigma_k^2)\right) + \lambda\left(\sum_{k=1}^{K} \pi_k - 1\right) \tag{80}$$

From the definition of r_{nk}, it is easy to see, that at each point n, r_{nk} will be proportional to the ratio of $\pi_k/(\sum\limits_{j=1}^{K} \pi_j)$. Summing over N will output the number of times class k is picked, and then summing over N will output N, the number of samples. Using this, the above is reformulated as

$$\sum_{k=1}^{K} N_k \sum_{i=1}^{N} log(\pi_k) + N log\left(\phi(y_i; w_k^T x_i + b_k, \sigma_k^2)\right) + \lambda\left(\sum_{k=1}^{K} \pi_k - 1\right) \tag{81}$$

Taking $\frac{\partial L}{\partial \pi_k}$, we get

$$\frac{N_k}{\pi_k} + \lambda = 0 \tag{82}$$

$$\pi_k = -\frac{N_k}{\lambda} \tag{83}$$

Substituting this back into the equation above, we see

$$\sum_{k=1}^{K} N_k \sum_{i=1}^{N} log\left(\frac{N_k}{\lambda}\right) + Nlog\left(\phi(y_i; w_k^T x_i + b_k, \sigma_k^2)\right) + \lambda\left(\sum_{k=1}^{K} \frac{N_k}{\lambda} - 1\right) \tag{84}$$

$$= \sum_{k=1}^{K} N_k \sum_{i=1}^{N} log\left(\frac{N_k}{\lambda}\right) + Nlog\left(\phi(y_i; w_k^T x_i + b_k, \sigma_k^2)\right) + \lambda\left(\sum_{k=1}^{K} N_k - \lambda\right) \tag{85}$$

Taking $\frac{\partial L}{\partial \lambda}$, we get

$$\sum_{k=1}^{K} N_k \frac{-N_k/\lambda^2}{N_k/\lambda} - 1 = 0 \tag{86}$$

$$-\frac{N}{\lambda} - 1 = 0 \tag{87}$$

$$\lambda = -N \tag{88}$$

and therefore $\pi_k = N_k/N$.

To solve for σ^2, we take $\partial Q/\partial \sigma^2$, treating σ^2 as its own variable. Using this, we get:

$$\frac{\partial}{\partial \sigma^2}\left(\sum_{i=1}^{N}\sum_{k=1}^{K} r_{nk} log\left(\frac{1}{\sqrt{2\pi\sigma^2}} exp\frac{-\|y_i - (w_k x_i + b_i)\|^2}{2\sigma^2}\right)\right) \tag{89}$$

$$= \left(\sum_{i=1}^{N}\sum_{k=1}^{K} r_{nk} \frac{(\|y_i - (w_k x_i + b_i)\|^2}{2\sigma^4} - \frac{1}{2\sigma^2}\right) \tag{90}$$

$$\Rightarrow \sigma^2 = \frac{\left(\sum_{i=1}^{N}\sum_{k=1}^{K} r_{nk}(\|y_i - (w_k x_i + b_i)\|^2\right)}{\left(\sum_{i=1}^{N}\sum_{k=1}^{K} r_{nk}\right)} \tag{91}$$

To solve for w and b I will solve the two sets of equations $\partial Q/\partial w$ and $\partial Q/\partial b$. Using the above equation, we find

$$\frac{\partial Q}{\partial w} = \sum_{i=1}^{N}\sum_{k=1}^{K} r_{nk}(y_i - (w_k x_i + b_i))x_i = 0 \tag{92}$$

$$\frac{\partial Q}{\partial b} = \sum_{i=1}^{N}\sum_{k=1}^{K} r_{nk}(y_i - (w_k x_i + b_i)) = 0 \tag{93}$$

Solving the first equation, we see

$$w_k = \frac{\sum_{i=1}^{N} r_{nk}(y_n - b_k)x_n}{\sum_{i=1}^{N} r_{nk} x_n x_n} \tag{94}$$

Substituting this into $\frac{\partial Q}{\partial b}$ and solving, we get

$$\sum_{i=1}^{N} r_{nk} y_n - r_{nk}\left[\frac{\sum_{i=1}^{N} r_{nk}(y_n x_n)}{\sum_{i=1}^{N} r_{nk} x_n x_n}\right] x_n = b_k\left(\sum_{i=1}^{N} r_{nk} - r_{nk}\left[\frac{\sum_{i=1}^{N} r_{nk} x_n}{\sum_{i=1}^{N} r_{nk} x_n x_n}\right] x_n\right) \tag{95}$$

or

$$b_k = \frac{\sum_{i=1}^{N} r_{nk} y_n - r_{nk} \left[\frac{\sum_{i=1}^{N} r_{nk}(y_n x_n)}{\sum_{i=1}^{N} r_{nk} x_n x_n} \right] x_n}{\left(\sum_{i=1}^{N} r_{nk} - r_{nk} \left[\frac{\sum_{i=1}^{N} r_{nk} x_n}{\sum_{i=1}^{N} r_{nk} x_n x_n} \right] x_n \right)} \tag{96}$$

And substituting this back into the equation for w_k, we can solve for both variables.

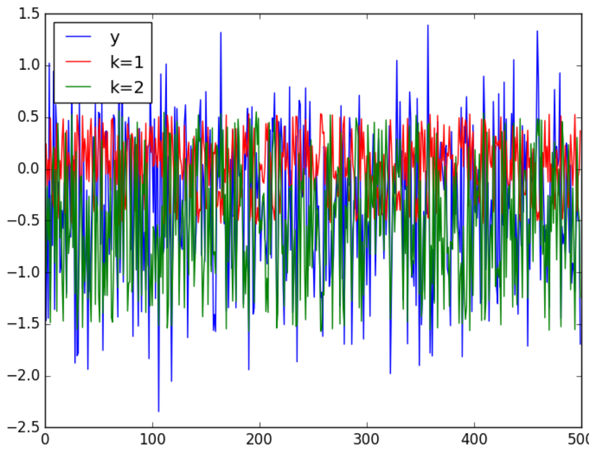
5.5 Part e

Shown is the code:

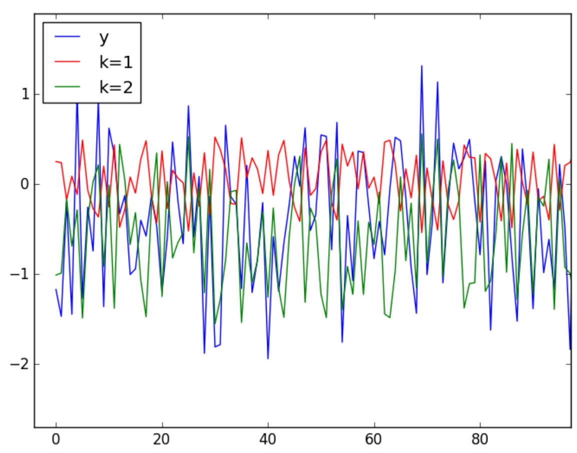
```
from __future__ import division
import numpy as np
from matplotlib import pyplot as plt
from scipy import stats

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

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

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

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


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

pih = np.array([0.5,0.5])
wh  = np.array([1,-1])
bh  = np.array([0,0])
sigmah  = (np.std(y)*np.array([1,1]))
r   = np.zeros((N,2))
Dist=np.zeros((N,2))
pi  = []
pi.append(pih)
p=0
LogL=[]
```

```
Q=0
Qold = 1
w=[]
b=[]
sigma=[]

while abs(Q-Qold)>10**(-4):
Qold = Q
print "hello"
for n in range(0,N):
sum=0
for j in range(0,2):
Mu     = wh[j]*x[n]+bh[j]
sum = sum+pih[j]*stats.norm(Mu,sigmah[j]).pdf(y[n])
Dist[n,j] = np.log(stats.norm(Mu,sigmah[j]).pdf(y[n]))
for k in range(0,2):
Mu     = wh[k]*x[n]+bh[k]
r[n,k] = pih[k]*stats.norm(Mu,sigmah[k]).pdf(y[n])/sum
print "bye"

    #---------------------E-STEP--------------------------%
Q = -np.sum(r*np.log(pih))+np.sum(r*Dist)
LogL.append(Q)

    #---------------------M-STEP--------------------------%
pih = r.sum(axis=0)/N
pi.append(pih)


yn    = np.transpose(np.array([y,y]))
xn    = np.transpose(np.array([x,x]))
Den   = (r*xn*xn).sum(axis=0)
LHS   = (r*yn).sum(axis=0)-((r*yn*xn).sum(axis=0)*xn*r/Den).sum(axis=0)
RHS   = r.sum(axis=0)-((r*xn).sum(axis=0)*r*xn/Den).sum(axis=0)

bh    =  LHS/RHS
b.append(bh)
wh     = (r*yn*xn-r*bh*xn).sum(axis=0)/Den
w.append(wh)

Mult2  = r*((yn-(xn*wh+bh))**2)
sigmah = np.sqrt(Mult2.sum(axis=0)/r.sum(axis=0))
sigma.append(sigmah)


print (pih,bh,sigmah,wh)
print Q-Qold
print "-------------------"
```

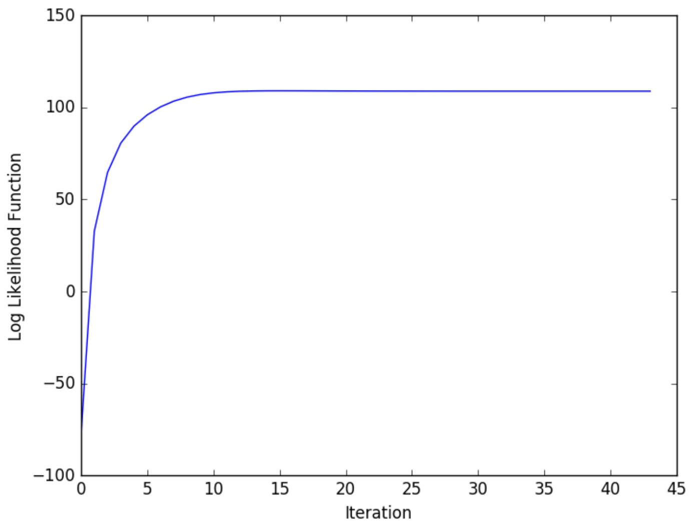
With the following deliverables

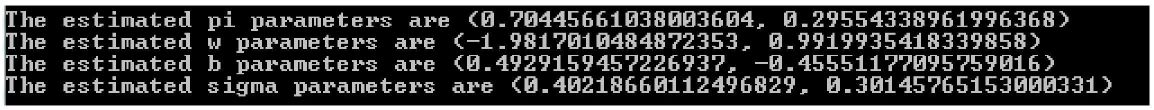


(a) Estimated Data vs Actual Data



(b) Zoomed in Estimated vs. Actual





```
The estimated pi parameters are (0.704456610380003604, 0.29554338961996368)
The estimated w parameters are (-1.9817010484872353, 0.99199354183339858)
The estimated b parameters are (0.4929159457226937, -0.4555117709575901 6)
The estimated sigma parameters are (0.4021866011249 6829, 0.3014576515300 0331)
```