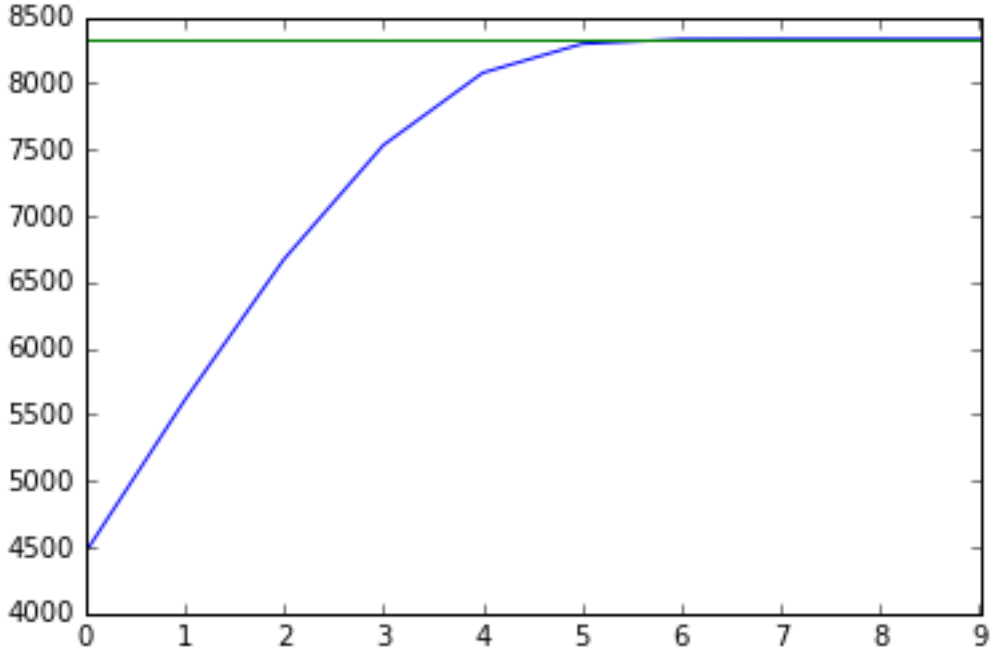
2.(f)

The plot of the log-likelihood vs iteration is as follows:

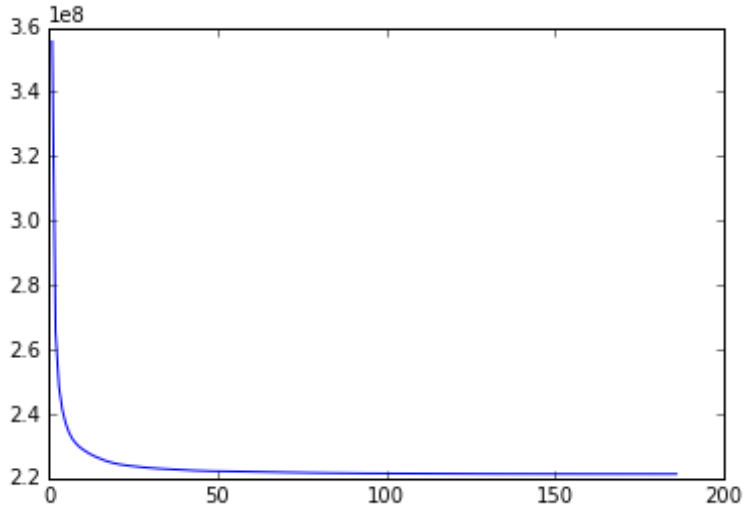


The estimated model parameters are:

9.96979256, 4.90985978, 14.86165346, 19.66495057, 48.97431779

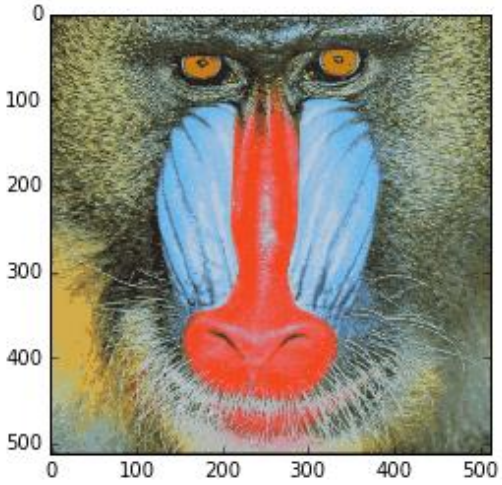
4.(a)

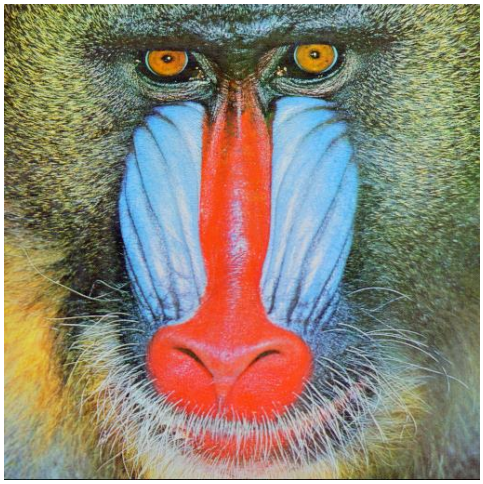
The plot of objective function is



Put the two pictures together:

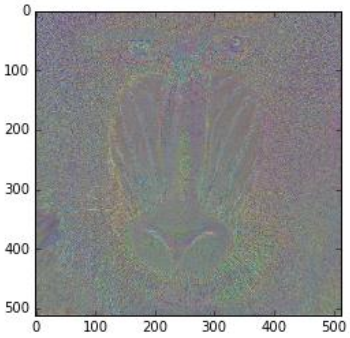
Compressed **Original**





Comparing the compressed picture with the original one, we can find the large parts with single color are best preserved. For the border parts where the color changes from one to another, they are generally not preserved well.

The picture of the difference:



The compression ratio is :

$$\mathrm{r} = \frac{24 * 2^2 * 64 + 216^2 * log_2 64}{24 * 512^2} = 6.3477\%$$

The relative mean absolute error of the compresses image is: 0.05

4.(b)

The number of 24 bits per pixel needed:

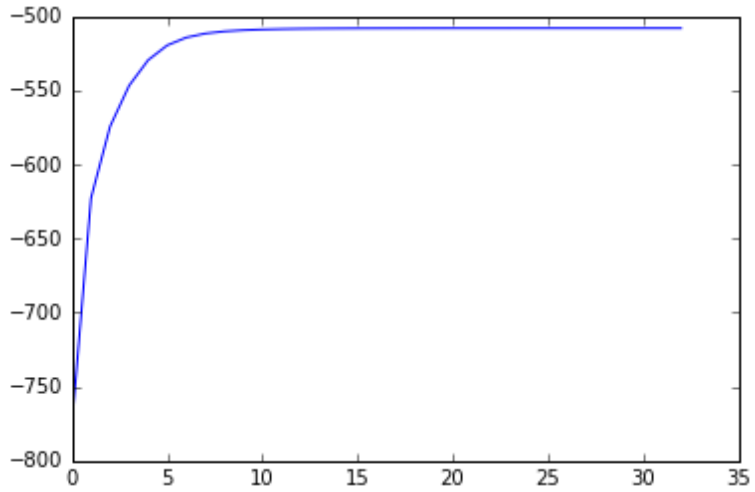
$$\mathrm{bpp} = \frac{24 * M^2 * K + \left(\frac{N}{M}\right)^2 * log_2 K}{N^2}$$

the compress ratio is:

$$\mathrm{ratio} = \mathrm{bpp}/24 = \frac{24 * M^2 * K + \left(\frac{N}{M}\right)^2 * log_2 K}{24N^2}$$

5.(e)

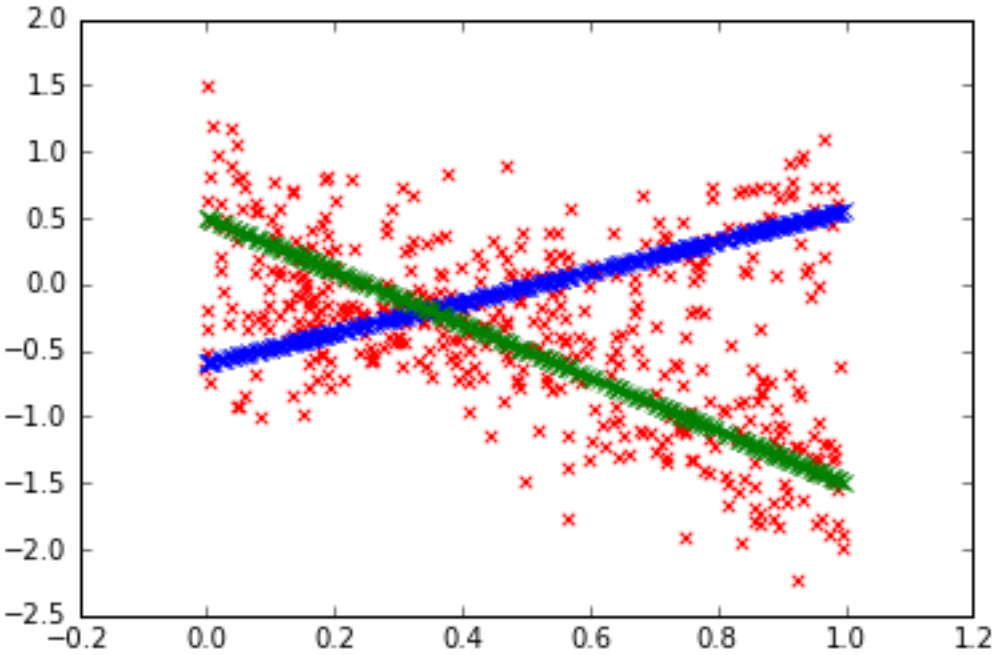
The plot of the log-likelihood is :



The estimated model parameters:

$$\pi = (0.293, 0.707), \quad (w, b) = \begin{pmatrix} 0.960 & -0.474 \\ -2.063 & 0.492 \end{pmatrix}, \quad \sigma^2 = 0.1778$$

The plot showing the data and estimated lines is :



Codes:

```
#2.f
import numpy as np
from matplotlib import pyplot as plt
from scipy.special import gammaln, polygamma
from __future__ import division
N = 1000
m = 5
alpha = np.array([10, 5, 15, 20, 50])
P = np.random.dirichlet(alpha, N)
t=np.sum(np.log(P),axis=0)/N
t=t.reshape((5,1))
U=np.ones((5,1))
alpha0=np.ones((5,1))
loglihd=np.zeros((1000,1))
for i in range(1000):
    dalpha1=N*(t+polygamma(0,np.sum(alpha0))*U-polygamma(0,alpha0));
    g1=-N*polygamma(1,alpha0)
    Q=np.diag(np.diag(np.ones((5,5))*g1))
    C=N*polygamma(1,np.sum(alpha0))
    Q1=np.linalg.inv(Q)
    alphan=alpha0-(Q1-
(Q1.dot(C*U.dot(U.T)).dot(Q1))/(1+C*U.T.dot(Q1).dot(U))).dot(dalpha1)
    loglihd[i]=N*((alphan-1).T.dot(t)+gammaln(np.sum(alphan))-
np.sum(gammaln(alphan)))
    if i==0:
        alpha0=alphan
    elif i > 0 and np.abs(loglihd[i]-loglihd[i-1])>0.0001:
        alpha0=alphan
    else:
        alphafnl=alphan
        break
lstd=N*((alpha-1).T.dot(t)+gammaln(np.sum(alpha))-np.sum(gammaln(alpha)))
x=np.linspace(0,9,10);y=loglihd[0:10];stdl=lstd*np.ones((10,1))
plt.plot(x,y);plt.plot(x,stdl)


#4.a
import numpy as np
from matplotlib import pyplot as plt
from __future__ import division
from scipy.ndimage import imread
mandrill = imread('mandrill.png', mode='RGB').astype(float)
N = int(mandrill.shape[0])
```

```
M = 2;k = 64
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)
Jf=np.zeros((300,1))
#calculating tagfet value J
def calcJ(data,centers):
    diffsq=(centers[:,np.newaxis,:]-data)**2
    return np.sum(np.min(np.sum(diffsq,axis=2),axis=0))
#implement k means
def kmeans(data,k):
#initializing centers and list J
    centers=data[np.random.choice(range(data.shape[0]),k,replace=False),:]
    J=[];
#closest center for each sample
    for itera in range(300):
        sqdistances=np.sum((centers[:,np.newaxis,:]-data)**2,axis=2)
        closest=np.argmin(sqdistances,axis=0)
#calculate J and append to list
        J.append(calcJ(data,centers))
        Jf[itera]=calcJ(data,centers)
#update clusters
        for i in range(k):
            centers[i,:]=data[closest==i,:].mean(axis=0)
#decide whether stopping
        if itera>0 and np.abs(Jf[itera]-Jf[itera-1])==0:
            X=centers[closest,:]
            break
        else:
            continue
    J.append(calcJ(data,centers))
    return J,centers,closest

JF,centersf,closestf=kmeans(X,64)
Xnew=centersf[closestf,:]
mandrillnew=np.zeros(512*512*3)
mandrillnew=mandrillnew.reshape(512,512,3)
for i in range(256):
    for j in range(256):
        mandrillnew[i*2:(i+1)*2,j*2:(j+1)*2,:]=Xnew[i*256+j,:].reshape(2,2,3)
plt.imshow(mandrillnew/255)
plt.show()
mandrillgrey=mandrillnew-mandrill+128*np.ones(512*512*3).reshape(512,512,3)
```

```
plt.imshow(mandrillgrey/255)
plt.show()
Jf1=Jf[Jf>0]
x1=np.linspace(1,186,186)
plt.plot(x1,Jf1)
error=np.sum(np.abs(mandrillnew-mandrill))/(3*255*512**2)


#5.e
from __future__ import division
import numpy as np
from matplotlib import pyplot as plt
from scipy.stats import norm as norm
# 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]

ccpiest = np.array([0.5, 0.5]);cwest = np.array([1, -1])
cbest = np.array([0, 0]);sigmaest = np.array([np.std(y), np.std(y)])
cwest2 = np.array([cwest,cbest]).T;x2 = np.array([x,np.ones(N)])
p1 = ccpiest[0]*norm(cwest2[0].dot(x2),sigmaest[0]).pdf(y)
p2 = ccpiest[1]*norm(cwest2[1].dot(x2),sigmaest[1]).pdf(y)
r1 = p1/(p1+p2);r2 = p2/(p1+p2);r = np.array([r1,r2])
Q1 = np.sum(r1*np.log(ccpiest[0]*norm(cwest2[0].dot(x2),sigmaest[0]).pdf(y)))
Q2 = np.sum(r2*np.log(ccpiest[1]*norm(cwest2[1].dot(x2),sigmaest[1]).pdf(y)))
Q = Q1+Q2;diff = 1;ll = [];ite = [];count = 0

for i in range(100):
    ll.append(Q);ite.append(count);tmp = Q
    ccpiest = np.array([np.mean(r1),np.mean(r2)])
    w1 = np.linalg.inv(x2.dot(np.diag(r1)).dot(x2.T)).dot(x2).dot(np.diag(r1)).dot(y)
    w2 = np.linalg.inv(x2.dot(np.diag(r2)).dot(x2.T)).dot(x2).dot(np.diag(r2)).dot(y)
    cwest2 = np.array([w1,w2])
```

```
        sigmaest1 = ((np.sum(r1*(y-cwest2[0].dot(x2))**2))/np.sum(r1))**0.5
        sigmaest2 = ((np.sum(r2*(y-cwest2[1].dot(x2))**2))/np.sum(r2))**0.5
        sigmaest = np.array([sigmaest1,sigmaest2])
        p1 = ccpiest[0]*norm(cwest2[0].dot(x2),sigmaest[0]).pdf(y)
        p2 = ccpiest[1]*norm(cwest2[1].dot(x2),sigmaest[1]).pdf(y)
        r1 = p1/(p1+p2);r2 = p2/(p1+p2);r = np.array([r1,r2])
        Q1 = np.sum(r1*np.log(ccpiest[0]*norm(cwest2[0].dot(x2),sigmaest[0]).pdf(y)))
        Q2 = np.sum(r2*np.log(ccpiest[1]*norm(cwest2[1].dot(x2),sigmaest[1]).pdf(y)))
        Q = Q1+Q2;diff = Q-tmp;count = count+1
        if diff>=1e-4:
            continue
        else:
            break
plt.plot(x, cwest2[0].dot(x2), c='b', marker='x')
plt.plot(x, cwest2[1].dot(x2), c='g', marker='x')
plt.scatter(x, y, c='r', marker='x')
plt.plot(ite,ll)
```