(1)

(a) $D_{KL}(p\|q) = D_{KL}(p(x,y)\|q(x,y))$

$$= \sum_x \sum_y p(x,y) \log \frac{p(x,y)}{q_1(x) q_2(y)}$$

$$= \sum_x \sum_y p(x,y) \log p(x,y) - \sum_x \sum_y p(x,y) \log q_1(x) q_2(y)$$

To minimize $D_{KL}(p\|q)$, we just need to minimize the right part.

$$- \sum_x \sum_y p(x,y) \log q_1(x) = - \sum_x \left[\left(\sum_y p(x,y)\right) \log q_1(x)\right] = - \sum_x p(x) \log q_1(x) = H(p,q)$$

As $H(p,q) \geq H(p)$, thus $\underset{q_1(x)}{\arg\min} - \sum_x \sum_y p(x,y) \log q_1(x) = p(x)$

Similarly, $\underset{q_2(y)}{\arg\min} - \sum_x \sum_y p(x,y) \log q_2(y) = p(y)$

Above all, the optimal approximation is a product of marginals.

(b) $D_{KL}(q\|p) = \sum_x \sum_y q(x,y) \log \frac{q(x,y)}{p(x,y)} = \sum_x \sum_y q_1(x) q_2(y) \log \frac{q_1(x) q_2(y)}{p(x,y)}$

For $p(x_i, y_i) = 0$, we need to set $q_1(x_i) q_2(y_i) = 0$ to avoid ∞ in $D_{KL}(q\|p)$, since $\lim_{x \to 0} x \log x = 0$. Therefore,

For $q_1(x_i) q_2(y_i) \log \frac{q_1(x_i) q_2(y_i)}{p(x_i, y_j)} \neq 0$, $\forall i, j$. Then there are 3 scenarios.

1. only $q_1(x_3), q_2(x_3) \neq 0$ 2. $q_1(x_4) q_2(x_4) \neq 0$ only.
3. $q_1(x_3), q_1(x_4), q_2(x_3), q_2(x_4) = 0$

Considering the general case: to minimize D_{KL},

s.t. $\sum_i q_1(x_i) = 1$, $\sum_j q_2(y_j) = 1$

Thus $L = \sum_{ij} q_1(x_i) q_2(y_j) \log \frac{q_1(x_i) q_2(y_j)}{p(x_i, y_j)} + \lambda\left(\sum_i q_1(x_i) - 1\right) + \beta\left(\sum_j q_2(y_j) - 1\right)$

$\forall i$, $\frac{\partial L}{\partial q_1(x_i)} = \log q_1(x_i) + \sum_j q_2(y_j) \log q_2(y_j) - \sum_j q_2(y_j) \log p(x_i, y_j) + 1 + \lambda = 0$

$\forall j$, $\frac{\partial L}{\partial q_2(y_j)} = \log q_2(y_j) + \sum_i q_1(x_i) \log q_1(x_i) - \sum_i q_1(x_i) \log p(x_i, y_j) + 1 + \beta = 0$

$\frac{\partial L}{\partial \lambda} = \sum_i q_1(x_i) = 0$, $\frac{\partial L}{\partial \beta} = \sum_j q_2(y_j) - 1 = 0$

For case (1), $q_1(x_3) = q_2(y_3) = 1$,

$$D_{KL} = 1 \cdot \log \frac{1}{\frac{1}{4}} = \log 4,$$

For case (2), $q_1(x_4) = q_2(y_4) = 1$,

$$D_{KL} = 1 \cdot \log \frac{1}{\frac{1}{4}} = \log 4$$

For case 3,

$$\begin{cases} \log q_1(x_1) + H(q_2) - \log \frac{1}{8} + 1 + \lambda = 0 \\ \log q_1(x_2) + H(q_2) - \log \frac{1}{8} + 1 + \lambda = 0 \\ \log q_2(y_1) + H(q_1) - \log \frac{1}{8} + 1 + \beta = 0 \\ \log q_2(y_2) + H(q_1) - \log \frac{1}{8} + 1 + \beta = 0 \\ q_2(y_1) + q_2(y_2) = 1 \\ q_1(x_1) + q_1(x_2) = 1 \end{cases} \Rightarrow q_1(x_1) = q_1(x_2) = q_2(y_1) = q_2(y_2) = 0.5$$

Thus $\min D_{KL}(q \| p) = \log 2$

(c) If we set $q(x,y) = p(x)p(y)$

$$\Rightarrow D_{KL} = \sum p(x_i) p(y_j) \log \frac{p(x_i) p(y_j)}{p(x_i, y_j)}$$

when $p(x_i, y_j) = 0$, $p(x_i) p(y_j) \neq 0$

$$\Rightarrow D_{KL}(q \| p) \to \infty$$

2. (a) $\Sigma = \begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{bmatrix}$.

Thus $\Sigma^{-1} = \begin{bmatrix} \frac{4}{3} & -\frac{2}{3} \\ -\frac{2}{3} & \frac{4}{3} \end{bmatrix}$, $|\Sigma| = \frac{3}{4}$

$$\therefore f(x_1, x_2) = \frac{1}{2\pi\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

$$(x-\mu)^T \Sigma^{-1} (x-\mu) = (x_1-1, x_2-1) \begin{pmatrix} \frac{4}{3} & -\frac{2}{3} \\ -\frac{2}{3} & \frac{4}{3} \end{pmatrix} \begin{pmatrix} x_1-1 \\ x_2-1 \end{pmatrix}$$

$$= \frac{4}{3}\left[(x_1-1)^2 - (x_1-1)(x_2-1) + (x_2-1)^2\right]$$

$$\therefore f(x_1, x_2) = \frac{1}{2\pi} \cdot \frac{1}{\sqrt{|\Sigma|}} \exp\left(-\frac{2}{3}\left[(x_1-1)^2 - (x_1-1)(x_2-1) + (x_2-1)^2\right]\right)$$

$$= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(x_1-1)^2\right) \cdot \frac{1}{\sqrt{2\pi}\sqrt{\frac{3}{4}}} \exp\left(-\frac{1}{2}\left(x_2-1-\frac{1}{2}(x_1-1)\right) \cdot \frac{4}{3}\left(x_2-1-\frac{1}{2}(x_1-1)\right)\right)$$

Thus $f_1(x_1) = \int f(x_1, x_2)\, dx_2 = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(x_1-1)^2\right)$

$$f(x_2|x_1) = \frac{f(x_1, x_2)}{f(x_1)} = \frac{1}{\sqrt{2\pi}\sqrt{\frac{3}{4}}} \exp\left(-\frac{1}{2}\left(x_2-1-\frac{1}{2}(x_1-1)\right)\frac{4}{3}\left(x_2-1-\frac{1}{2}(x_1-1)\right)\right)$$

In the similar way,

$$f(x_1|x_2) = \frac{f(x_1, x_2)}{f(x_2)} = \frac{1}{\sqrt{2\pi}\sqrt{\frac{3}{4}}} \exp\left(-\frac{1}{2}\left(x_1-\frac{1}{2}x_2-\frac{1}{2}\right) \cdot \frac{4}{3} \cdot \left(x_1-\frac{1}{2}x_2-\frac{1}{2}\right)\right)$$

5. (a) $\because \check{X}$ is white

$\therefore \frac{1}{N} \check{X}\check{X}^T = I$

As $\check{X} = DX \Rightarrow \frac{1}{N} DXX^T D^T = I$

Since XX^T is PSD, thus we can use eigenvalue decomposition for XX^T, i.e. we can write

$XX^T = EVE^T$, E is the orthogonal matrix of eigenvectors of XX^T

V is the diagonal matrix of its eigenvalue $V = diag(V_1, \cdots V_n)$

Let $V^{\frac{1}{2}} = diag(V_1^{\frac{1}{2}}, V_2^{\frac{1}{2}}, \cdots V_n^{\frac{1}{2}})$

$$\therefore \frac{1}{N} DXX^T D^T = \frac{1}{N} DEV^{\frac{1}{2}}V^{\frac{1}{2}}E^T D^T$$

$$= \frac{1}{N} DEV^{\frac{1}{2}}E^{-1}EV^{\frac{1}{2}}E^T D^T$$

$$= \frac{1}{N} D(EV^{\frac{1}{2}}E^T)(EV^{\frac{1}{2}}E^T)^T D^T$$

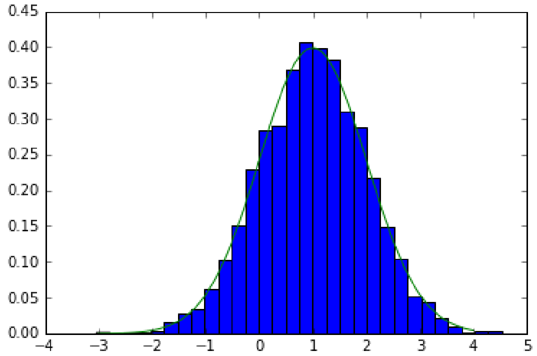
$$\therefore D = \sqrt{N} EV^{-\frac{1}{2}}E^T$$

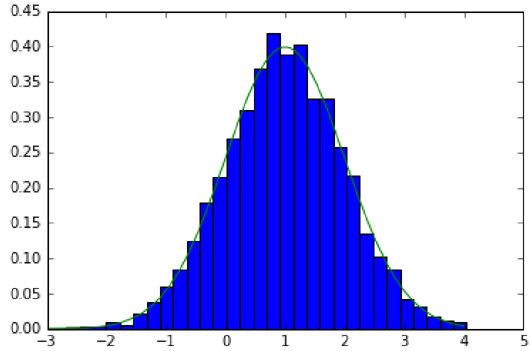
$$\frac{1}{N} DXX^T D^T = \frac{1}{N} \cdot N \cdot (EV^{-\frac{1}{2}}E^T)(EV^{\frac{1}{2}}E^T)(EV^{\frac{1}{2}}E^T)^T(EV^{-\frac{1}{2}}E^T)^T$$

$= I.$ white.

2.

#B



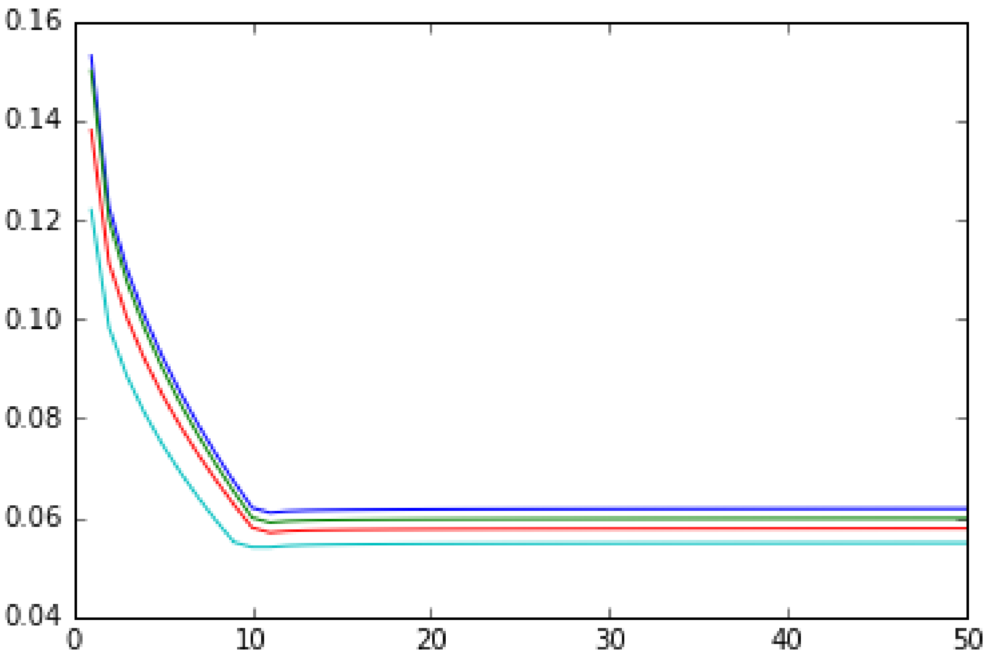


3.

#a

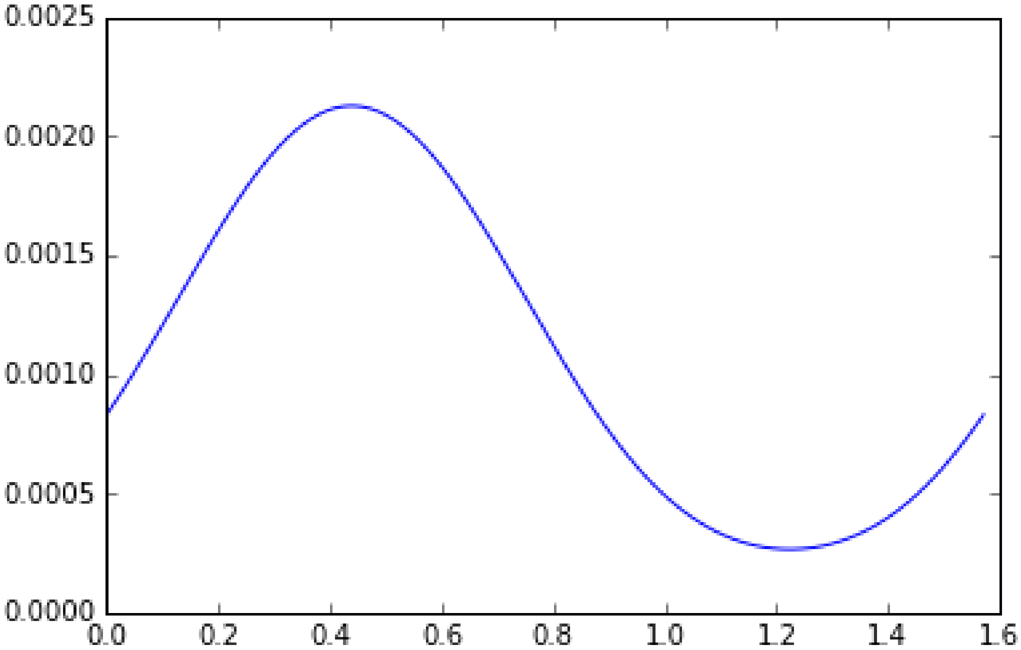
Sequence	Prior	Likelihood	Posterior
0,2,0,1	0.012	0.288	0.068
0,1,2,2	0.020	0.180	0.071
0,2,2,2	0.037	0.100	0.074

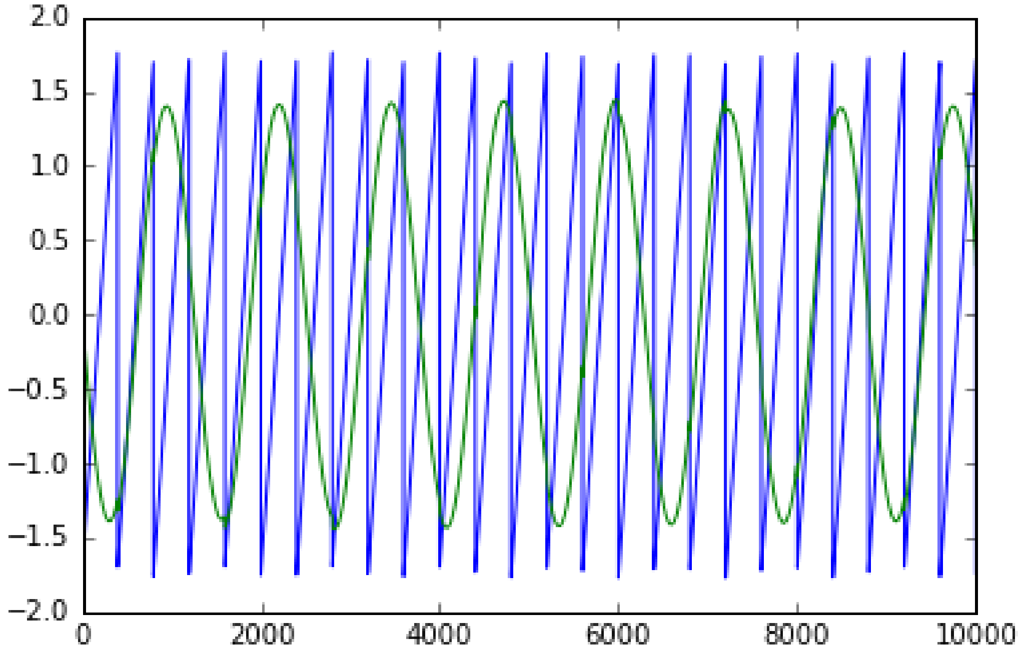
#b



From the highest line to the lowest line, N=5000,2000,1000,500

5.


0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
2000
4000
6000
8000
10000

Codes:

2.

```
#b
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.mlab as mlab
from __future__ import division
x1=np.zeros((5001,1));x2=np.zeros((5001,1))
for i in range (5000):
    x2[i]=np.sqrt(3)*np.random.randn()/2+0.5+0.5*x1[i]
    x1[i+1]=np.sqrt(3)*np.random.randn()/2+0.5+0.5*x2[i]
x1=x1[0:5000];x2=x2[0:5000];
m1=np.average(x1[4000:5000]);m2=np.average(x2[4000:5000])
x=np.linspace(-4,4,100)
plt.hist(x1[4000:5000],20,normed='TRUE')
plt.plot(x,mlab.normpdf(x,m1,np.sqrt(3)/2))
plt.hist(x2[4000:5000],20,normed='TRUE')
plt.plot(x,mlab.normpdf(x,m1,np.sqrt(3)/2))
```

3

```
#a
import numpy as np
from matplotlib import pyplot as plt
zf=np.zeros((81,4))
for i in range(81):
    for j in range(4):
        zf[i,j]=(i/3**j)%3
p=np.zeros((81,1))
from __future__ import division
A=np.matrix(([0.5,0.2,0.3],[0.2,0.4,0.4],[0.4,0.1,0.5]))
phi=np.matrix(([0.8,0.2],[0.1,0.9],[0.5,0.5]))
pi0=np.matrix(([0.5],[0.3],[0.2]))
for i in range(81):
    p[i]=pi0[zf[i,0]]*A[zf[i,0],zf[i,1]]*A[zf[i,1],zf[i,2]]*A[zf[i,2],zf[i,3]]
    p[i]=p[i]*phi[zf[i,0],0]*phi[zf[i,1],1]*phi[zf[i,2],0]*phi[zf[i,3],1]
(p.T).argsort()[0][-3:]
px=np.sum(p);fv=np.zeros((100,3));
for i in [33,75,78]:
    fv[i,0]=pi0[zf[i,0]]*A[zf[i,0],zf[i,1]]*A[zf[i,1],zf[i,2]]*A[zf[i,2],zf[i,3]]
    fv[i,1]=p[i]/fv[i,0]
    fv[i,2]=p[i]/px


#b
```

```
import numpy as np
import math
from matplotlib import pyplot as plt
zf=np.zeros((81,4))
for i in range(81):
    for j in range(4):
        zf[i,j]=(i/3**j)%3
from __future__ import division
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])
X = []
for _ in xrange(5000):
    z = [np.random.choice([0,1,2], p=pi0)]
    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]
    X.append(x)

def fb_alg(A_mat, O_mat, observ):
    # set up
    k =int(observ.size/4)
    (n,m) = O_mat.shape
    prob_mat = np.zeros( (n,k) )
    fw = np.zeros( (n,k+1) )
    bw = np.zeros( (n,k+1) )
    # forward part
    fw[:, 0] = 1.0/n
    for obs_ind in xrange(k):
        f_row_vec = np.matrix(fw[:,obs_ind])
        fw[:, obs_ind+1] = f_row_vec * \
                                   np.matrix(A_mat) * \
                                   (np.matrix(np.diag(O_mat[:,observ[obs_ind]])).T
        fw[:,obs_ind+1] = fw[:,obs_ind+1]/np.sum(fw[:,obs_ind+1])
    # backward part
    bw[:,-1] = 1.0
    for obs_ind in xrange(k, 0, -1):
        b_col_vec = np.matrix(bw[:,obs_ind]).T
        bw[:, obs_ind-1] = ((np.matrix(np.diag(O_mat[:,observ[obs_ind-1]]))) * \
                                   (np.matrix(A_mat)).T * \
                                    b_col_vec).T
        bw[:,obs_ind-1] = bw[:,obs_ind-1]/np.sum(bw[:,obs_ind-1])
    # combine it
    prob_mat = np.array(fw)*np.array(bw)
```

```
        prob_mat = prob_mat/np.sum(prob_mat, 0)
        # get out
        return prob_mat, fw, bw

#main function
def baum_welch( num_states, num_obs, observ ):
        # allocate
        A_mat = np.ones( (num_states, num_states) )
        A_mat = (A_mat.T / np.sum(A_mat,1)).T
        O_mat = np.ones( (num_states, num_obs) )
        O_mat = (O_mat.T / np.sum(O_mat,1)).T
        theta = np.zeros( (num_states, num_states, observ.size) )
        sig=np.zeros(50)
        for iter in range(50):
                old_A = A_mat
                old_O = O_mat
                A_mat = np.ones( (num_states, num_states) )
                O_mat = np.ones( (num_states, num_obs) )
                # expectation step, forward and backward probs
                P,F,B = fb_alg( old_A, old_O, observ)
                # need to get transitional probabilities at each time step too
                px=np.zeros((16,81))
                for j in range(16):
                        for i in range(81):
                                px[j,i]=
pi0[zf[i,0]]*old_A[zf[i,0],zf[i,1]]*old_A[zf[i,1],zf[i,2]]*old_A[zf[i,2],zf[i,3]]

px[j,i]=px[j,i]*old_O[zf[i,0],xf[j,0]]*old_O[zf[i,1],xf[j,1]]*old_O[zf[i,2],xf[j,2]]*old_O[zf[i,3],xf

[j,3]]
                px=np.sum(px,1)
                sig[iter]=np.sum(px-px0)/2
                for a_ind in xrange(num_states):
                        for b_ind in xrange(num_states):
                                for t_ind in xrange(observ.size):
                                        theta[a_ind,b_ind,t_ind] = \
                                        F[a_ind,t_ind] * \
                                        B[b_ind,t_ind+1] * \
                                        old_A[a_ind,b_ind] * \
                                        old_O[b_ind, observ[t_ind]]
                # form A_mat and O_mat
                for a_ind in xrange(num_states):
                        for b_ind in xrange(num_states):
                                A_mat[a_ind, b_ind] = np.sum( theta[a_ind, b_ind, :] )/ \
```

```
                                                np.sum(P[a_ind,:])
        A_mat = A_mat / np.sum(A_mat,1)
        for a_ind in xrange(num_states):
            for o_ind in xrange(num_obs):
                right_obs_ind = np.array(np.where(observ == o_ind))+1
                O_mat[a_ind, o_ind] = np.sum(P[a_ind,right_obs_ind])/ \
                                                np.sum( P[a_ind,1:])
        O_mat = O_mat / np.sum(O_mat,1)
        # compare
        if np.linalg.norm(old_A-A_mat) < .00001 and np.linalg.norm(old_O-O_mat) < .00001:
            break
    # get out
    return sig

#use the train data
xf=np.zeros((16,4))
for i in range(16):
    for j in range(4):
        xf[i,j]=(i/2**j)%2
px0=np.zeros((16,81))
for j in range(16):
    for i in range(81):
        px0[j,i]= pi0[zf[i,0]]*A[zf[i,0],zf[i,1]]*A[zf[i,1],zf[i,2]]*A[zf[i,2],zf[i,3]]
        px0[j,i]=px0[j,i]*phi[zf[i,0],xf[j,0]]*phi[zf[i,1],xf[j,1]]*phi[zf[i,2],xf[j,2]]*phi[zf[i,3],xf[j,3]]
px0=np.sum(px0,1)
test500=np.array(X[0:500]);test1500=np.array(X[0:1500]);
test1000=np.array(X[0:1000]);test2000=np.array(X[0:2000]);
sig500=baum_welch(3,2,test500);sig1500=baum_welch(3,2,test1500);
sig1000=baum_welch(3,2,test1000);sig2000=baum_welch(3,2,test2000);
xplot=np.linspace(1,51,50)
plt.plot(xplot,sig500);plt.plot(xplot,sig1500);
plt.plot(xplot,sig1000);plt.plot(xplot,sig2000);
```

4.

```
import numpy as np
from scipy import stats
import matplotlib.cm as cm
from sklearn import datasets, linear_model
from matplotlib import pyplot as plt;
train_noised = np.genfromtxt('train_noised.csv',delimiter=',',skip_header=1)
train_noised = train_noised.transpose()
train_noised = train_noised[1:].transpose()
train_clean = np.genfromtxt('train_clean.csv',delimiter=',',skip_header=1)
train_clean = train_clean.transpose()
train_clean = train_clean[1:].transpose()
test_noised = np.genfromtxt('test_noised.csv',delimiter=',',skip_header=1)
test_noised = test_noised.transpose()
test_noised = test_noised[1:].transpose()
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
trainx = get_patches(train_noised)
trainy = train_clean.reshape((392000,1))
testx = get_patches(test_noised)
#slope, intercept, r_value, p_value, std_err = stats.linregress(trainx,trainy)
rgs = linear_model.LinearRegression()
rgs.fit(trainx,trainy)
result = rgs.predict(testx)
for i in range(78400):
    if result[i]<0:
        result[i]=0
    if result[i]>255:
        result[i]=255
import csv
with open('myres.csv', 'wb') as f:
    writer = csv.writer(f)
    writer.writerows(result)
```

5.

```
#b
from __future__ import division
import numpy as np
from matplotlib import pyplot as plt
N = 10000
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)
V,E=np.linalg.eig(np.dot(X,X.T))
V1=1/np.sqrt(V);V1=np.diag(V1);
D=np.sqrt(N)*np.dot(np.dot(E,V1),E.T)
X1=np.dot(D,X);th=np.zeros((2000,1));J=np.zeros((2000,1))
for i in range(2000):
    th[i]=i*0.5*(np.pi)/1999;w=np.zeros((2,2));
    w[0,0]=np.cos(th[i]);w[0,1]=-np.sin(th[i]);
    w[1,0]=np.sin(th[i]);w[1,1]=np.cos(th[i]);
    J[i]=np.sum((np.mean(G(np.dot(w.T,X1)),1)-gamma)**2)
plt.plot(th,J)
thf=np.argmax(J)*0.5*(np.pi)/1999;wf=np.zeros((2,2));
wf[0,0]=np.cos(thf);wf[0,1]=-np.sin(thf);
wf[1,0]=np.sin(thf);wf[1,1]=np.cos(thf);
yf=np.dot(wf.T,X1)
plt.plot(th,J)
plt.plot(yf[0,:])
plt.plot(yf[1,:])
```