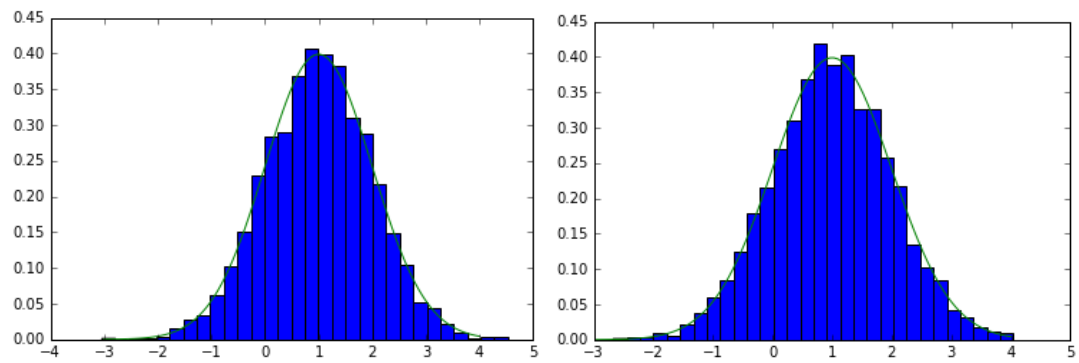


2.

#B

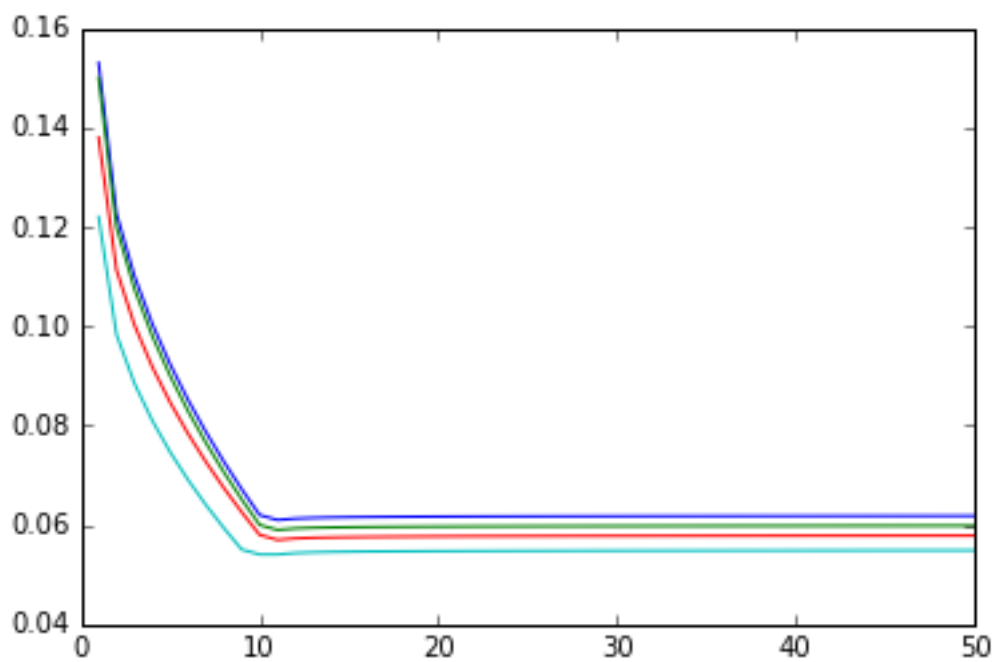


3.

#a

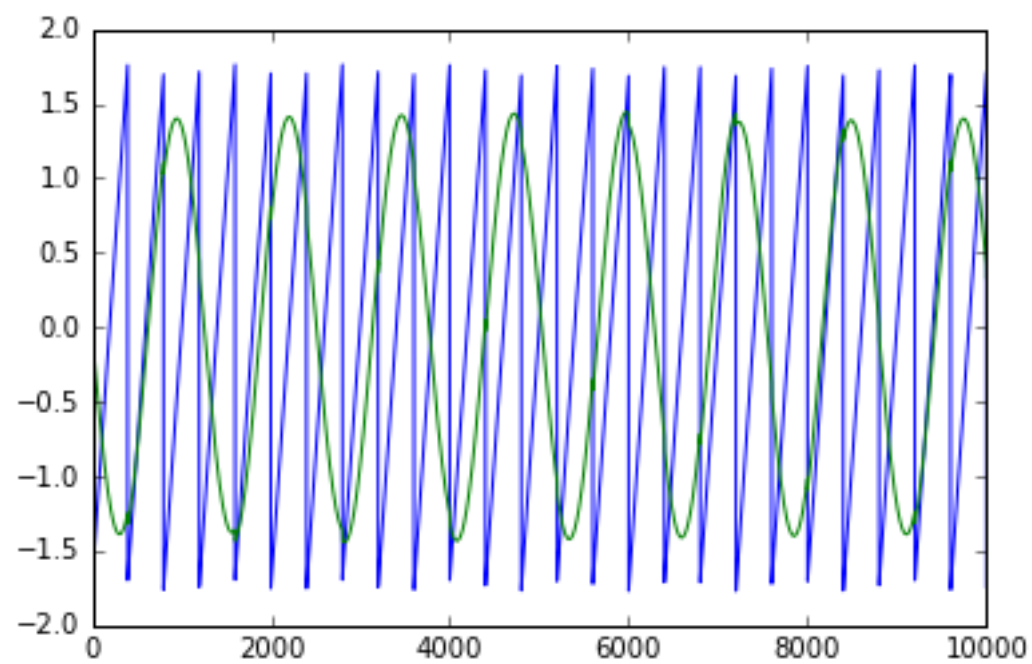
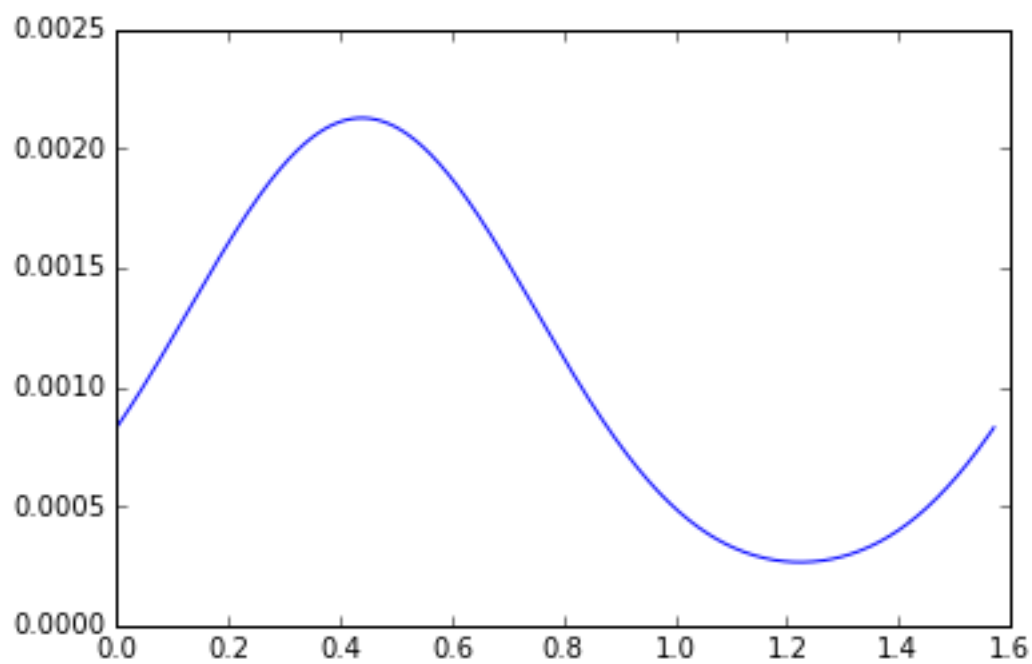
| Sequence | Prior | Likelihood | Posterior |
|----------|-------|------------|-----------|
| 0,2,0,1 | 0.012 | 0.288 | 0.068 |
| 0,1,2,2 | 0.020 | 0.180 | 0.071 |
| 0,2,2,2 | 0.037 | 0.100 | 0.074 |

#b



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

5.



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,:])
```