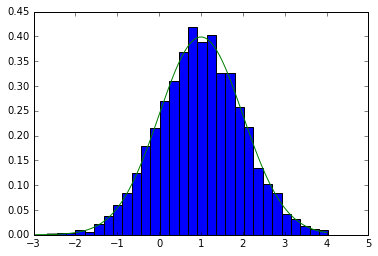
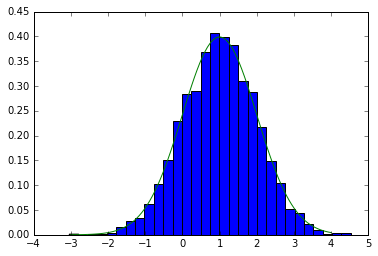
2.

#B

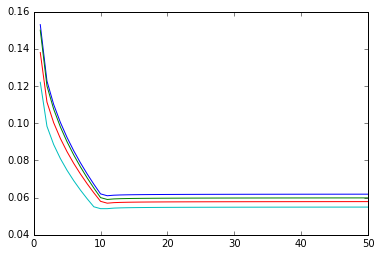


3.

#a

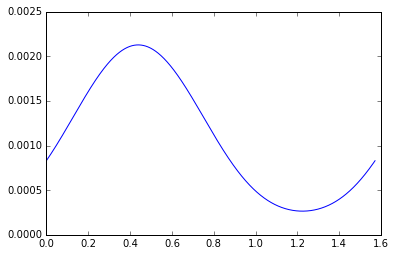
|  |  |  |  |
| --- | --- | --- | --- |
| Sequence | Prior | Likelihood | Posterior |
| 0,2,0,1 | 0.012 | 0.288 | 0.068 |
| 0,1,2,2 | 0.020 | 0.180 | 0.071 |
| 0,2,2,2 | 0.037 | 0.100 | 0.074 |

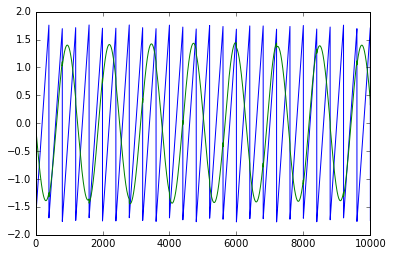
#b



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,:])