ECE 5412 Final Exam Yucheng Chen

Question 1.

(a). The signals sent by two channels can be modeled as a 2-state Markov chain. The measurement of the signals are corrupted by additive a Gaussian noise and a sinusoidal bias.

(b). Suppose the posterior of estimated state at time step is

The transition probability .

The emission probability

Given the posterior of estimated state at time , the transition probability and emission probability The posterior of estimated state at time is computed as follows:

.

(c). The plot of conditional mean estimate is shown as the following plot:



MATLAB code:

clc,clear,close

P = [0.8,0.2;0.2,0.8];

pi0 = [0.1,0.9];

tspan = 1000;

y = zeros(1,tspan);

s\_true = zeros(1,tspan);

pi\_d = pi0;

A = 0.5;

omega = 5;

%genenrate true state sequence and observation vector (y)

for i = 1:tspan

s = find(mnrnd(1,pi\_d)) - 2;

if s == 0

s = 1;

end

s\_true(i) = s;

y(i) = s + randn + A\*sin(omega\*i);

pi\_d = pi\_d\*P;

pi\_d = pi\_d / sum(pi\_d);

end

p\_prv = 0.5;

p\_nxt = p\_prv;

s\_est = zeros(1,tspan);

%implementation of filter

for i = 1:tspan

p\_nxt = normpdf(y(i),-1 + A\*sin(omega\*i),1)\*(p\_prv \* P(1,1) + (1 - p\_prv)\* P(2,1))/(normpdf(y(i),-1 + A\*sin(omega\*i),1)\*(p\_prv \* P(1,1) + (1 - p\_prv)\* P(2,1)) + ...

normpdf(y(i),1 + A\*sin(omega\*i),1)\*(p\_prv \* P(1,2) + (1 - p\_prv)\* P(2,2)));

s\_est(i) = -p\_nxt + (1 - p\_nxt);

p\_prv = p\_nxt;

end

plot(1:tspan,s\_true,'+',1:tspan,s\_est,'\*');

title('True state vs Estimated State');

legend('True State','Condition Mean estimate');

xlabel('Time Steps')

(d). Through 50 independent simulations, the mean squared error of the state estimate is

The distribution of the 50 simulations is shown in the following plot:



MATLAB code:

clc,clear,close

P = [0.8,0.2;0.2,0.8];

pi0 = [0.1,0.9];

tspan = 1000;

num\_sim = 50;

A = 0.5;

omega = 5;

errors = zeros(1,num\_sim);

for j = 1:num\_sim

pi\_d = pi0;

y = zeros(1,tspan);

s\_true = zeros(1,tspan);

%genenrate true state sequence and observation vector (y)

for i = 1:tspan

s = find(mnrnd(1,pi\_d)) - 2;

if s == 0

s = 1;

end

s\_true(i) = s;

y(i) = s + randn + A\*sin(omega\*i);

pi\_d = pi\_d\*P;

pi\_d = pi\_d / sum(pi\_d);

end

p\_prv = 0.5;

p\_nxt = p\_prv;

s\_est = zeros(1,tspan);

%implementation of filter

for i = 1:tspan

p\_nxt = normpdf(y(i),-1 + A\*sin(omega\*i),1)\*(p\_prv \* P(1,1) + (1 - p\_prv)\* P(2,1))/(normpdf(y(i),-1 + A\*sin(omega\*i),1)\*(p\_prv \* P(1,1) + (1 - p\_prv)\* P(2,1)) + ...

normpdf(y(i),1 + A\*sin(omega\*i),1)\*(p\_prv \* P(1,2) + (1 - p\_prv)\* P(2,2)));

s\_est(i) = -p\_nxt + (1 - p\_nxt);

p\_prv = p\_nxt;

end

errors(j) = norm(s\_true - s\_est);

end

plot(1:num\_sim,errors,'\*')

xlabel('Sim trials')

ylabel('Error')

(e). Given the observation sequence

where can be computed through optimal filter.

Assumptions: All conditional probability models are time invariant and all parameters are known.

Denote . Consider we have a 2-state Markov chain,

Then the recursion is:

Thus, given ,

Denote un-normalized distribution at time step k as and normalized distribution as .

The smooth algorithm is as follows:

for state at each time step:

(f). The errors of 50 simulations are shown in the figure below and the mean squared error is 0.5141, which is less than the mean squared error of the filter-based state estimation:



MATLAB code:

clc,clear,close

P = [0.8,0.2;0.2,0.8];

pi0 = [0.1,0.9];

tspan = 1000;

num\_sim = 50;

A = 0.5;

omega = 5;

errors = zeros(1,num\_sim);

errors\_filter = zeros(1,num\_sim);

for j = 1:num\_sim

pi\_d = pi0;

y = zeros(1,tspan);

s\_true = zeros(1,tspan);

%genenrate true state sequence and observation vector (y)

for i = 1:tspan

s = find(mnrnd(1,pi\_d)) - 2;

if s == 0

s = 1;

end

s\_true(i) = s;

y(i) = s + randn + A\*sin(omega\*i);

pi\_d = pi\_d\*P;

pi\_d = pi\_d / sum(pi\_d);

end

p\_nxt = 0.5;

s\_est = zeros(1,tspan);

s\_est\_filter = zeros(1,tspan);

%implementation of filter

for i = 1:tspan

%forward filter

p\_nxt = normpdf(y(i),-1 + A\*sin(omega\*i),1)\*(p\_nxt \* P(1,1) + (1 - p\_nxt)\* P(2,1))/(normpdf(y(i),-1 + A\*sin(omega\*i),1)\*(p\_nxt \* P(1,1) + (1 - p\_nxt)\* P(2,1)) + ...

normpdf(y(i),1 + A\*sin(omega\*i),1)\*(p\_nxt \* P(1,2) + (1 - p\_nxt)\* P(2,2)));

p\_back = [1;1];

%backward algorithm

for k = tspan:-1:i

B = diag([normpdf(y(k),-1 + A\*sin(omega\*k),1),normpdf(y(k),1 + A\*sin(omega\*k),1)]);

p\_back = P\*B\*p\_back;

p\_back = p\_back/sum(p\_back);

end

gamma1 = p\_nxt\*p\_back(1);

gamma2 = (1 - p\_nxt) \* p\_back(2);

gammas = [gamma1, gamma2]/(gamma1 + gamma2);

s\_est(i) = -gammas(1) + gammas(2);

s\_est\_filter(i) = -p\_nxt + (1 - p\_nxt);

end

errors(j) = norm(s\_true - s\_est);

errors\_filter(j) = norm(s\_true - s\_est\_filter);

end

plot(1:num\_sim,errors,'\*')

xlabel('Sim trials')

ylabel('Error Smoother')

figure

plot(1:num\_sim,errors\_filter,'\*')

xlabel('Sim trials')

ylabel('Error filter')

(g). Derivation of ML estimator using EM algorithm for amplitude

Then the auxiliary likelihood is as follows:

where:

,

Then the EM algorithm goes as follows:

E step:

M step:

Compute the derivative of with respect to .

(h). We evaluate the log likelihood of the dataset given a parameter :

Run the EM algorithm and use to generate data and the final estimate of the parameter is .

The time history of log likelihood is shown as follows:



MATLAB code:

%question 1 EM

clc,clear,close

P = [0.8,0.2;0.2,0.8];

pi0 = [0.1,0.9];

tspan = 1000;

num\_sim = 50;

A = 0.5;

omega = 5;

pi\_d = pi0;

y = zeros(1,tspan);

s\_true = zeros(1,tspan);

%genenrate true state sequence and observation vector (y)

for i = 1:tspan

s = find(mnrnd(1,pi\_d)) - 2;

if s == 0

s = 1;

end

s\_true(i) = s;

y(i) = s + randn + A\*sin(omega\*i);

pi\_d = pi\_d\*P;

pi\_d = pi\_d / sum(pi\_d);

end

%EM algorithm

A = 10;

num\_iter = 50;

Logliks = zeros(1,num\_iter);

for i = 1:num\_iter%iterate 100 times

% E step

[gamma\_marginal, gamma\_joint] = computePosterior(y, A, tspan);

Logliks(i) = evalLoglik(gamma\_marginal, gamma\_joint, A, omega, tspan, y);

% M step

A = sum((y - (-gamma\_marginal + (1 - gamma\_marginal))).\*sin(omega\*(1:tspan)))/sum(sin(omega\*(1:tspan)).\*sin(omega\*(1:tspan)));

end

plot(1:num\_iter, Logliks)

function [gamma\_marginal, gamma\_joint] = computePosterior(y, A, tspan)

forward\_prob = zeros(1,tspan);

backward\_prob = zeros(1,tspan);

gamma\_marginal = zeros(1,tspan);%prob of being -1

gamma\_joint = zeros(4,tspan-1);%(-1,-1), (-1,1), (1,-1), (1,1)

P = [0.8,0.2;0.2,0.8];

omega = 5;

p\_back = [1;1];

%backward algorithm

for k = tspan:-1:1

B = diag([normpdf(y(k),-1 + A\*sin(omega\*k),1),normpdf(y(k),1 + A\*sin(omega\*k),1)]);

p\_back = P\*B\*p\_back;

p\_back = p\_back/sum(p\_back);

backward\_prob(k) = p\_back(1);

end

p\_nxt = 0.5;

for i = 1:tspan

%forward filter

p\_nxt = normpdf(y(i),-1 + A\*sin(omega\*i),1)\*(p\_nxt \* P(1,1) + (1 - p\_nxt)\* P(2,1))/(normpdf(y(i),-1 + A\*sin(omega\*i),1)\*(p\_nxt \* P(1,1) + (1 - p\_nxt)\* P(2,1)) + ...

normpdf(y(i),1 + A\*sin(omega\*i),1)\*(p\_nxt \* P(1,2) + (1 - p\_nxt)\* P(2,2)));

gamma1 = p\_nxt\*backward\_prob(i);

gamma2 = (1 - p\_nxt) \* (1 - backward\_prob(i));

gammas = [gamma1, gamma2]/(gamma1 + gamma2);

gamma\_marginal(i) = gammas(1);

forward\_prob(i) = p\_nxt;

end

for i = 1:tspan-1

%s\_{i-1} = -1, s\_i = -1

gamma\_joint(1,i) = forward\_prob(i)\*P(1,1)\*normpdf(y(i+1),-1 + A\*sin(omega\*(i+1)),1)\*backward\_prob(i+1);

%s\_{i-1} = -1, s\_i = 1

gamma\_joint(2,i) = forward\_prob(i)\*P(1,2)\*normpdf(y(i+1),1 + A\*sin(omega\*(i+1)),1)\*(1 - backward\_prob(i+1));

%s\_{i-1} = 1, s\_i = -1

gamma\_joint(3,i) = (1 - forward\_prob(i))\*P(2,1)\*normpdf(y(i+1),-1 + A\*sin(omega\*(i+1)),1)\* backward\_prob(i+1);

%s\_{i-1} = 1, s\_i = 1

gamma\_joint(4,i) = (1 - forward\_prob(i))\*P(2,2)\*normpdf(y(i+1),1 + A\*sin(omega\*(i+1)),1)\* (1 - backward\_prob(i+1));

gamma\_joint(:,i) = gamma\_joint(:,i)/sum(gamma\_joint(:,i));

end

end

function Loglik = evalLoglik(gamma\_marginal, gamma\_joint, A, omega, tspan, y)

P = [0.8,0.2;0.2,0.8];

Loglik = -sum(gamma\_marginal/2.\*(y + 1 - A\*sin(omega\*(1:tspan))).^2) - sum((1 - gamma\_marginal)/2.\*(y - 1 - A\*sin(omega\*(1:tspan))).^2);

for i = 1:size(gamma\_joint,1)

gamma = gamma\_joint(:,i);

Loglik = Loglik + gamma(1)\*log(P(1,1)) + gamma(2)\*log(P(1,2)) + gamma(3)\*log(P(2,1)) + gamma(4)\*log(P(2,2));

end

end

Question 2.

(a). Using MH algorithm to simulate the distribution, the histogram of the data points is shown as follows:



The empirical and the true CDFs are shown as follows:



MATLAB code:

clc,clear,close all

%question 2a

numSamples = 100000;

propVar = 5;

x = 1;

x\_arr = zeros(1,numSamples);

for i = 1:numSamples

x\_cand = normrnd(x,propVar);

x\_prob = cos(x)^2\*sin(2\*x)^2\*normpdf(x);

x\_cand\_prob = cos(x\_cand)^2\*sin(2\*x\_cand)^2\*normpdf(x\_cand);

if x\_cand\_prob > x\_prob

x = x\_cand;

else

u = rand;

if u < x\_cand\_prob/x\_prob

x = x\_cand;

end

end

x\_arr(i) = x;

end

step\_cdf = 0.01;

lower\_bound = -4;

upper\_bound = 4;

cdf = [];

cum\_den = 0;

for i = lower\_bound:step\_cdf:upper\_bound

cum\_den = cum\_den + cos(i)^2\*sin(2\*i)^2\*normpdf(i)\*step\_cdf;

cdf = [cdf,cum\_den];

end

cdf = cdf/max(cdf);

figure

[f,x\_h] = ecdf(x\_arr);

plot(x\_h,f,'b','LineWidth',2);

hold on

plot(lower\_bound:step\_cdf:upper\_bound,cdf,'r--','LineWidth',2)

legend('Empirical CDF','True CDF');

(b). Although the measurement noise is non-Gaussian, we assume it is Gaussian with the same variance when we develop Kalman filter.

Given , where is the variance of the true measurement noise.

The equations of the Kalman filter is as follows:

Prediction:

Update:

(c). Simulate the Kalman filter on the state space model. The comparison of the true state and the state estimate is as follows:



The mean squared error is .

MATLAB code:

clc,clear,close all

%question 2c

numSamples = 200000;

propVar = 5;

x = 1;

x\_arr = zeros(1,numSamples);

for i = 1:numSamples

x\_cand = normrnd(x,propVar);

x\_prob = cos(x)^2\*sin(2\*x)^2\*normpdf(x);

x\_cand\_prob = cos(x\_cand)^2\*sin(2\*x\_cand)^2\*normpdf(x\_cand);

if x\_cand\_prob > x\_prob

x = x\_cand;

else

u = rand;

if u < x\_cand\_prob/x\_prob

x = x\_cand;

end

end

x\_arr(i) = x;

end

%state is z,

%generate state and observation data

z = 1;

tspan = 500;

z\_arr = zeros(1,tspan);

y\_arr = zeros(1,tspan);

for i = 1:tspan

z = z + normrnd(0,1);

y = z + x\_arr(randi(numSamples,1));

z\_arr(i) = z;

y\_arr(i) = y;

end

R = var(x\_arr);

%Kalman filter

xhat = 0;

Sigma = 1;

x\_est\_arr = zeros(1,tspan);

for i = 1:tspan

xhat\_pred = xhat;

Sigma\_pred = Sigma + 1;

S = Sigma\_pred + R;

xhat = xhat\_pred + Sigma\_pred\*(y\_arr(i) - xhat\_pred)/S;

Sigma = Sigma\_pred - Sigma\_pred^2/S;

x\_est\_arr(i) = xhat;

end

plot(1:tspan,z\_arr,1:tspan,x\_est\_arr)

legend('True State','State Estimate')

xlabel('Time')

(d). The Kalman filter is NOT optimal in terms of mean squared error because the observation noise is not Gaussian.

(e). The comparison of true state, Kalman filter estimate and particle filter estimate is shown as the following plot:



The mean squared error of Kalman filter is and the mean squared error of particle filter is

MATLAB code:

%particle filter

particalSize = 2000;

zhat = 10\*rand(particalSize,1);

x\_est\_particle = zeros(1,tspan);

for i = 1:tspan

zhat\_pred = zhat + normrnd(0,1,particalSize,1);

w = cos(y\_arr(i)-zhat\_pred).^2.\*sin(2\*(y\_arr(i)-zhat\_pred)).^2.\*normpdf(y\_arr(i)-zhat\_pred);

partition = mnrnd(particalSize,w/sum(w));

zhat = [];

for j = 1:length(partition)

num = partition(j);

for k = 1:num

zhat = [zhat,zhat\_pred(j)];

end

end

x\_est\_particle (i) = mean(zhat);

end

figure

plot(1:tspan,z\_arr,1:tspan,x\_est\_arr)

hold on

plot(1:tspan,x\_est\_particle)

xlabel('Time')

legend('True State','Kalman', 'Particle');