**Appendix (Matlab Code)**

**A.1 Kalman Filter Model**

%%%Kalman filter model%%%

clear;

%Work on training data to estimate parameters

load midterm\_train;

[M, C] = size(rate); %Row and column dimensions

%Centralize the data

mean\_pos = mean(kin(:,1:2)); %Only need the positions (first 2 components of x)

%ones(M,1) is M by 1 and mean\_pos is 1 by 2

kin(:,1:2) = kin(:,1:2) - ones(M,1)\*mean\_pos;

mean\_rate = mean(rate);

%# of rows of rate = # of rows of kin

rate = rate - ones(M,1)\*mean\_rate;

% Model identification

a = kin(2:M,:)';

b = kin(1:M-1,:)';

c = rate';

d = kin';

%The definition of a, b, c and d smartly manipulates matrix multiplication

%in the closed-form formulae

%The estimates are denoted as Ah, Wh, Hh, and Qh

Ah = a\*b'\*inv(b\*b');

Wh = (a-Ah\*b)\*(a-Ah\*b)'/(M-1);

Hh = c\*d'\*inv(d\*d');

Qh = (c-Hh\*d)\*(c-Hh\*d)'/M;

%Neural decoding on test data by Kalman filter algorithm

%Loading test data has to be done here and we need to redefine dimensions

load midterm\_test;

[M, C] = size(rate);

%C actually equals 42

%Here we use the mean values of the training data

%since we "pretend" not to know kin in the test data

rate = rate - ones(M,1)\*mean\_rate;

d = size(kin,2);

%Define initial values of all estimates

x\_minus = zeros(d, M);

x = zeros(d, M);

P\_minus = zeros(d, d, M);

P = zeros(d, d, M);

K = zeros(d, C, M);

for k = 2:M

% prior estimation

P\_minus(:,:,k) = Ah\*P(:,:,k-1)\*Ah'+Wh;

x\_minus(:,k) = Ah\*x(:,k-1);

% posterior estimation

K(:,:,k) = P\_minus(:,:,k)\*Hh'\*inv(Hh\*P\_minus(:,:,k)\*Hh'+Qh);

P(:,:,k) = (eye(d)-K(:,:,k)\*Hh)\*P\_minus(:,:,k);

x(:,k) = x\_minus(:,k)+K(:,:,k)\*(rate(k,:)'-Hh\*x\_minus(:,k));

end

%Centralization "recovery"

x(1:2,:) = x(1:2,:) + mean\_pos'\*ones(1,M);

x\_minus(1:2,:) = x\_minus(1:2,:) + mean\_pos'\*ones(1,M);

%Show the decoding of x and y positions

figure(1);

subplot(2,1,1);

plot(1:M, kin(:,1),'r--','linewidth', 2);

hold on;

plot(1:M, x(1,:)','b--','linewidth',2);

ylabel('x-position','FontSize', 10);

legend('True','Reconstructed','Location','best');

s1=subplot(2,1,1);

title(s1,'Reconstruction of KF Algorithm, x-position', 'Fontsize', 12);

subplot(2,1,2);

plot(1:M, kin(:,2),'r--','linewidth', 2);

hold on;

plot(1:M, x(2,:)','b--','linewidth',2);

ylabel('y-position','FontSize', 10);

legend('True','Reconstructed','Location','best');

s2=subplot(2,1,2);

title(s2,'Reconstruction of KF Algorithm, y-position', 'Fontsize', 12);

xlabel('time (t)');

% R^2 Error

r2 = 1 - sum((x(1:2,:)'-kin(:,1:2)).^2)./ ...

sum((kin(:,1:2)-ones(M,1)\*mean\_pos).^2);

disp(['R2 = ' num2str(r2)]);

% Sequential Monte Carlo algorithm

K = 20; % sample size, subject to change

% initialize the sample data set

s = zeros(d,K,M);

weight = ones(K,M)/K;

Weight = cumsum(weight); %Accumulative weights

x\_h = zeros(d,M);

for t = 2:M

if mod(t,100) == 0

disp(sprintf('t = %d', t));

end

r = rand(1,K);

j = sum(Weight(:,t-1)\*ones(1,K) < ones(K,1)\*r)+1;

s\_p = s(:,j,t-1);

s(:,:,t) = Ah\*s\_p + mvnrnd(zeros(1,d),Wh,K)';

%compute the (log) likelihood

for k = 1:K

log\_like(k,1) = -0.5\*(log(det(Qh))+(rate(t,:)'-Hh\*s(:,k,t))'...

\*inv(Qh)\*(rate(t,:)'-Hh\*s(:,k,t)));

end

% the weights at step t

weight(:,t) = ones(K,1)./sum(exp(ones(K,1)\*log\_like'-log\_like\*ones(1,K)),2);

Weight(:,t) = cumsum(weight(:,t));

% estimate the kinematics

x\_h(:,t) = s(:,:,t)\*weight(:,t);

end

% centralization "recovery"

x\_h(1:2,:) = x\_h(1:2,:) + mean\_pos'\*ones(1,M);

% show the decoding of x and y positions

figure(2);

subplot(2,1,1);

plot(1:M, kin(:,1),'r--','linewidth', 2);

hold on;

plot(1:M, x\_h(1,:)','b--','linewidth', 2);

ylabel('x-position','FontSize', 10);

legend('True','Reconstructed','Location','best');

s1=subplot(2,1,1);

title(s1,'Reconstruction of SMC Algorithm, x-position (K=20)', 'Fontsize', 12);

subplot(2,1,2);

plot(1:M, kin(:,2),'r--','linewidth', 2);

hold on;

plot(1:M, x\_h(2,:)','b--','linewidth', 2);

ylabel('y-position','FontSize', 10);

legend('True','Reconstructed','Location','best');

s2=subplot(2,1,2);

title(s2,'Reconstruction of SMC Algorithm, y-position (K=20)', 'Fontsize', 12);

xlabel('time (t)');

%R^2 Error

r2 = 1 - sum((x\_h(1:2,:)'-kin(:,1:2)).^2) ./ ...

sum((kin(:,1:2)-ones(M,1)\*mean\_pos).^2);

disp(['R2 = ' num2str(r2)]);

**A.2 Inhomogeneous Poisson Process Model**

% Inhomogeneous Poisson process

clear;

%Work on training data to estimate parameters

load midterm\_train;

[M, C] = size(rate); %Row and column dimensions

d = size(kin,2);

%Centralize the data

kin\_mean = mean(kin);

kin = kin - ones(M,1)\*kin\_mean;

% Fit the kinematic model v\_{k+1} = A\*v\_k+w\_k

% Ah, Wh are the same as in a Kalman filter model

a = kin(2:M,:)';

b = kin(1:M-1,:)';

Ah = a\*b'\*inv(b\*b');

Wh = (a-Ah\*b)\*(a-Ah\*b)'/(M-1);

% Estimate mu\_c and alpha\_c using MLE method

for c = 1:C

alpha\_old = zeros(d+1,1);

alpha\_new = alpha\_old + 1;

% Newton-Raphson method, closed-form formulae in lecture notes

count(c) = 0;

while norm(alpha\_new-alpha\_old) > 1e-2

count(c) = count(c) + 1;

alpha\_old = alpha\_new;

Kin = [ones(M,1) kin];

nmt = Kin'\*rate(:,c) - Kin'\*exp(Kin\*alpha\_old);

dnt = - Kin'.\*(ones(d+1,1)\*exp(Kin\*alpha\_old)')\*Kin;

alpha\_new = alpha\_old - dnt\nmt;

end

alpha(:,c) = alpha\_new;

end

%Decoding using Point Process filter on the test data

load midterm\_test;

M2 = size(rate, 1);

%Set initial kinematic (x) values

x\_m(:,1) = zeros(d,1);

x(:,1) = x\_m(:,1);

P\_m(:,:,1) = zeros(d);

P(:,:,1) = zeros(d);

%Reconstruction by PPF

for k = 2:M2

P\_m(:,:,k) = Ah\*P(:,:,k-1)\*Ah'+Wh;

x\_m(:,k) = Ah\*x(:,k-1);

P(:,:,k) = inv(inv(P\_m(:,:,k)) + ...

alpha(2:end,:)\*diag(exp([1 x\_m(:,k)']\*alpha))\*alpha(2:end,:)');

x(:,k) = x\_m(:,k)+P(:,:,k)\*(alpha(2:end,:)\*(rate(k,:)-exp([1 x\_m(:,k)']\*alpha))');

end

%Centralization "recovery"

x = x + kin\_mean'\*ones(1,M2);

% R^2 Error

r2 = 1 - sum((x'-kin).^2)./ ...

sum((kin-ones(M2,1)\*kin\_mean).^2);

disp(['R2 = ' num2str(r2)]);

figure(1);

%Here d=4, so it's good to use a loop

for i = 1:d

subplot(2,2,i);

plot(1:M2,kin(:,i)','r--','linewidth', 2);

hold on;

plot(1:M2, x(i,:),'b--','linewidth', 2);

legend('True','Reconstructed','Location','best');

s=subplot(2,2,i);

title(s,'Reconstruction of PPF Algorithm', 'Fontsize', 12);

end

%Reconstruction by SMC

load midterm\_test;

M2 = size(rate, 1);

K = 500; %sample size, subject to change

%Assign initial values to the sample data set

s = zeros(d,K,M2);

weight = ones(K,M2)/K;

Weight = cumsum(weight);

x\_h = zeros(d,M2);

for t = 2:M2

if mod(t,100) == 0

disp(sprintf('t = %d', t));

end

r = rand(1,K);

j = sum(Weight(:,t-1)\*ones(1,K) < ones(K,1)\*r)+1;

s\_p = s(:,j,t-1);

s(:,:,t) = Ah\*s\_p + mvnrnd(zeros(1,d),Wh,K)';

%Compute the (log) likelihood

for k = 1:K

lambda = exp(alpha'\*[1; s(:,k,t)]);

log\_like(k,1) = sum(log(poisspdf(rate(t,:)', lambda)));

end

% the weights at step t

weight(:,t) = ones(K,1)./sum(exp(ones(K,1)\*log\_like'-log\_like\*ones(1,K)),2);

Weight(:,t) = cumsum(weight(:,t));

% estimate the kinematic (x) values

x\_h(:,t) = s(:,:,t)\*weight(:,t);

end

%Centralization "recovery"

x\_h = x\_h + kin\_mean'\*ones(1,M2);

%R^2 Error

r2 = 1 - sum((x\_h'-kin).^2)./ ...

sum((kin-ones(M2,1)\*kin\_mean).^2);

disp(['R2 = ' num2str(r2)]);

%Visualization display

figure(2);

for i = 1:d

subplot(2,2,i);

plot(1:M2,kin(:,i)','r--','linewidth', 2);

hold on;

plot(1:M2, x\_h(i,:),'b--','linewidth', 2);

legend('True','Reconstructed','Location','best');

s=subplot(2,2,i);

title(s,'Reconstruction of SMCM, K=500', 'Fontsize', 12);

end