HW 13 Code

T = 200;

t = 1:T;

F = [0.5, 0.5, 0; 0, 0.5, 0.5; 0, 0, 0.5]; %Transition matrix for state model

H = [1, 1, 1]; %Measurement matrix

Q = [1, 0, 0; 0, 1, 0; 0, 0, 1]; %Covariance Matrix for state-model random noise w

R = 1; %Covariance Matrix for measurement noise/error

xtrue = zeros(3,T); %true states of x

xtrue = [0; 0; 0]; %Our initial guess/estimate for x

xtrue = zeros(3, T);

xtrue = [0; 0; 0];

wk = randn([3 T], 'like', T);

for k = 2:T

xtrue(:, k) = F\*xtrue(:, k-1) + w(:, k-1);

end

v = randn(1,T);

z = H\*xtrue + v;

xhat = zeros(3,T);

xhat(:, 1) = [0; 0; 0];

P = [1, 0, 0; 0, 1, 0; 0, 0, 1]; %Error Covariance at time 0

I3 = eye(3);

for k = 2:T

xhat(:, k) = F\*xhat(:, k-1); %One step prediction

P = F\*P\*F' + Q; %Error Covariance update

K = P\*H'/(H\*P\*H' + R); %Gain matrix

xhat(:, k) = xhat(:, k) + K\*(z(k) - H\*xhat(:,k)); %update for estimate at time k

P = (I3 - K\*H)\*P;

end

function meas(k, x(k), v(k))

    H = [1, 1, 1];

    for k = 2:T

        z(:,k)= H\*x(:,k) + v(:,k);

    end

end

function state(k, x, w)

    F = [0.5, 0.5, 0; 0, 0.5, 0.5; 0, 0, 0.5];

    for k = 2:T

        x(:,k)= F\*x(:,k-1) + w(:,k-1);

    end

end

N = 10;

T = 200;

nx = 3;

nz = 1;

x = zeros(nx,T);

x0 = zeros(nx,1); % initial states

z = zeros(nz,T);

F = [0.5, 0.5, 0; 0, 0.5, 0.5; 0, 0, 0.5];

H = [1, 1, 1];

Q = [1, 0, 0; 0, 1, 0; 0, 0, 1];

R = 1;

for k = 2:T

state(:,k) = F\*x(:,k-1) + w(:,k-1);

end

w = mvnrnd([0,0,0]', Q, T)';

v = mvnrnd(0, R, T)';

for k = 2:T

state(:,k) = F\*x(:,k-1) + w(:,k-1);

end

for k = 2:T

meas(k) = H\*x(:,k) + v(k);

end

for k = 2:T

p\_xk\_given\_xk\_1(:,k) = mvnpdf(x(:,k-1), F\*x(:,k-1), Q);

end

p\_zk\_given\_xk = zeros(nz, T);

for k = 2:T

p\_zk\_given\_xk(k) = mvnpdf(z(k), H\*x(:,k), R);

end

% N: number of particles

% T: number of time

% nx: dimension of state x

% nz: dimension of measurement z

% state: state equation function

% meas: measurement equation function

% p\_zk\_given\_xk: probability of zk given xk

% state\_noise: distribution of state noise wk

% w: state noise wk

% v: measurement noise vk

% particles0: initial particles according to prior guess of true state

% If resampling is needed, then resampling = 1; otherwise resampling = 0.

function [x,xpf,alpha] = particle(N,T,nx,nz,state,meas,p\_zk\_given\_xk,state\_noise,w,v,particles0,resampling)

%% inital parameters set up

x = zeros(nx,T);

x0 = zeros(nx,1);  % initial states

z = zeros(nz,T);

alpha = zeros(N,T);

alpha0 = 1/N\*ones(N,1);  % initial weights

particles = zeros(nx,N,T);

x(:,1) = state(1, x0, w(:,1)); % calculate true state through state equation

z(:,1) = meas(1, x(:,1), v(:,1));

for i = 1 : N

    particles(:,i,1) = state(1, particles0(:,i), state\_noise(0));

    alpha(i,1) = alpha0(i) \* p\_zk\_given\_xk(1, z(:,1), particles(:,i,1));

end

alpha(:,1) = alpha(:,1)./sum(alpha(:,1));

%% Update particles and corresponding weights

for time = 2 : T

    x(:,time) = state(time, x(:,time-1), w(:,time));   % simulate state

    z(:,time) = meas(time, x(:,time), v(:,time));     % simulate measurement

    for i =  1 : N

        particles(:,i,time) = state(time, particles(:,i,time-1), state\_noise(0)); % update particles

        alpha(i,time) = alpha(i,time-1) \* p\_zk\_given\_xk(time, z(:,time), particles(:,i,time)); % update weights

    end

    alpha(:,time) = alpha(:,time)./sum(alpha(:,time));

    if resampling == 1

        Neff = 1/sum(alpha(:,time).^2);

        if Neff < 0.5\*N

            [particles(:,:,time), alpha(:,time)] = resample(particles(:,:,time),alpha(:,time));

        end

    end

end

%% Calculate the estimated states

xpf = zeros(nx,T);

for time = 1 : T

    for i = 1 : N

       xpf(:,time) = xpf(:,time) + alpha(i,time)\*particles(:,i,time);

    end

end

end

%% Resampling

function [particles,alpha]=resample(particles,alpha)

N = length(alpha);

res = randsample(1:N, N, true, alpha);

particles = particles(:,res);       % new particles

alpha = repmat(1/N, 1, N);          % now all particles have the same weight

end

particle(N,T,nx,nz,state,meas,p\_zk\_given\_xk,state\_noise,w,v,particles0,1);

figure(1);

plot(t, xhat(1,:), '-', 'LineWidth', 2);

hold on;

plot(t, xtrue(1,:), '-', 'LineWidth', 1.5);

hold on;

plot(t, state(1, :), '-', 'LineWidth', 1.5);

xlabel('time'); ylabel('x1 (First Component)');

grid on;

legend('x1 Estimated', 'x1 True');

figure(2);

plot(t, xhat(2,:), '-', 'LineWidth', 2);

hold on;

plot(t, x(2,:), '-', 'LineWidth', 1.5);

hold on;

plot(t, state(2, :), '-', 'LineWidth', 1.5);

xlabel('time'); ylabel('x2 (Second Component)');

grid on;

legend('x2 Estimate', 'x2 True');

figure(3);

plot(t, xhat(3,:), '-', 'LineWidth', 2);

hold on;

plot(t, x(3, :), '-', 'LineWidth', 1.5);

hold on;

plot(t, state(3, :), '-', 'LineWidth', 1.5)

xlabel('time'); ylabel('x3 (Third Component');

grid on;

legend('x3 Estimate', 'x3 True');

figure(4);

plot(t, xhat(1,:), '-', 'LineWidth', 2);

hold on;

plot(t, x(1,:), '-', 'LineWidth', 1.5);

hold on;

plot(t, state(1, :), '-', 'LineWidth', 1.5);

xlabel('time'); ylabel('x1 (First Component)');

grid on;

legend('x1 Estimated', 'x1 True');

figure(5);

plot(t, xhat(2,:), '-', 'LineWidth', 2);

hold on;

plot(t, x(2,:), '-', 'LineWidth', 1.5);

hold on;

plot(t, state(2, :), '-', 'LineWidth', 1.5);

xlabel('time'); ylabel('x2 (Second Component)');

grid on;

legend('x2 Estimate', 'x2 True');

figure(6);

plot(t, xhat(3,:), '-', 'LineWidth', 2);

hold on;

plot(t, x(3, :), '-', 'LineWidth', 1.5);

hold on;

plot(t, state(3, :), '-', 'LineWidth', 1.5);

xlabel('time'); ylabel('x3 (Third Component');

grid on;

legend('x3 Estimate', 'x3 True');