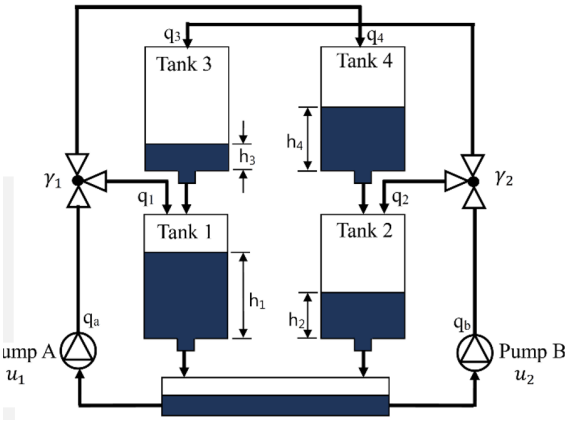
CH5120: Course Project 1

CH20B065 Mihir Singh

Part - 2 : Particle Filter

Problem Statement

The quadruple tank process consists of four interconnected water tanks. Given below is the schematic diagram. The equations representing the dynamics of the system are given as follows:-



$$\frac{dh_1}{dt} = -\frac{a_1}{A_1}\sqrt{2gh_1} + \frac{a_3}{A_1}\sqrt{2gh_3} + \frac{\gamma_1 k_1}{A_1}v_1$$
$$\frac{dh_2}{dt} = -\frac{a_2}{A_2}\sqrt{2gh_2} + \frac{a_4}{A_2}\sqrt{2gh_4} + \frac{\gamma_2 k_2}{A_2}v_2$$
$$\frac{dh_3}{dt} = -\frac{a_3}{A_3}\sqrt{2gh_3} + \frac{(1-\gamma_2)k_2}{A_3}v_2$$
$$\frac{dh_4}{dt} = -\frac{a_4}{A_4}\sqrt{2gh_4} + \frac{(1-\gamma_1)k_1}{A_4}v_1$$

where

A_i	cross-section of Tank i;
a_i	cross-section of the outlet hole;
h_i	water level.

Data Generation:

Solve the set of non-linear equations of the four-tank system, as given below, using forward difference method or by using ODE-45 of *Matlab* (or equivalent in *Python,* etc.), for generating a typical data set of 10,000. This data set is to be used for comparing the performance of the Kalman Filter & Particle Filter.

Particle Filter Implementation

Formulate a Particle Filter for the above four tank problem for estimating h-3 & h-4 (not measured) and obtaining the filtered values for h-1 & h-2 (measured). Verify the resulting estimate by comparing the same with the generated data set.

Introduction:-

The key difference between the kalman filter and the particle filter is that particle filters work for non-linear and non-gaussian models while kalman filters do not. For the purposes of implementation in this case, however, the assumption of a gaussian noise is still retained. The only difference is that we can now use the non-linear model described by the differential equations shown earlier to propagate the state and obtain the priors.

Approach:-

- First, the differential system of equations governing the four tanks system is simulated to generate the data that will be used to model the particle filters. The system of equations was simulated from 0 to 1000s at intervals of 0.1s.
- Next, the parameters of the particle filter system are initialized. These parameters include the covariance matrix of the noise in the state estimation and measurements, Q and R. The values of Q and R are set as 0.00001*eye(4) and 0.00001*eye(2). The initial posterior estimate is also roughened with gaussian noise to better emulate the real world.

```
%Initialization

x0 = h0;

Ts = tspan(2)-tspan(1);

n = 4; %Number of states

P0 = 0.0001*eye(4);%Initial Covariance Matrix

N = 400; %Number of particles

L = chol(P0);

x0 = (x0*ones(1,N)) + L*rand(n,N); %Adding covariance to each particle
```

```
x_post = x0;

%Roughing the posterior with Noise

w = chol(Q)*rand(n,N);

w1 = w(1,:);

w2 = w(2,:);

w3 = w(3,:);

w4 = w(4,:);

x_post(1,:) = x_post(1,:) + w1;

x_post(2,:) = x_post(2,:) + w2;

x_post(3,:) = x_post(3,:) + w3;

x_post(4,:) = x_post(4,:) + w4;
```

- The next step is to begin the loop of the particle filters. The loop is run for 10000 time instants. In every iteration, the state at the next instant is calculated using the equations used to define the dynamic system. Gaussian noise is added to this estimate as well. After this the importance weights are calculated to resample the particles. These resampled particles form the next set of 'posterior' values.

```
k=1;

while(k<=10000)

w = chol(Q)*randn(n,N); %for roughening the prior

x_dt = zeros(4,N);

%Prediction

x_dt(1,:) = (-a1/A1*sqrt(2*g*(x_post(1,:))) + a3/A1*sqrt(2*g*(x_post(3,:))) +
(gamma1*k1*v1)/A1);

x_dt(2,:) = -a2/A2*sqrt(2*g*(x_post (2,:))) + a4/A2*sqrt (2*g*(x_post(4,:))) +
(gamma2*k1*v2)/A2 ;

x_dt(3,:) = -a3/A3*sqrt (2*g*(x_post(3,:))) + (1 - gamma2)*k2*v2/A3;

x_dt(4,:) = -a4/A4*sqrt (2*g*(x_post(4,:))) + (1 - gamma1)*k1*v1/A4;

x_prior = x_post + x_dt*Ts + w;

%Importance weights
```

```
z_true = y(:,k) * ones(1,N);

z_est = C*x_prior;

v = z_true-z_est;

q = zeros(1,N);

wt = zeros(1,N);

for i=1:N

q(i) = exp(-0.5 * v(:,i)'*inv(R)*v(:,i));

end

for i=1:N

wt(i) = q(i)/sum(q);

end

j=1;

cumsum_wt = cumsum(wt);

for i=1:N

r = rand;

while cumsum_wt(j) < r

j=j+1;

end

x_post(:,i) = x_prior(:,j);

end
```

- At the end of every loop, the mean state of all the particles is recorded.

The complete code base is provided below:

```
clc;

clear;
```

```
close all;

%Initial Values

A1 = 28; %(cm^2)

A2 = 32;

A3 = 28;

A4 = 32;

a1 = 0.071; a3 = 0.071; %(cm^2)

a2 = 0.057; a4 = 0.057;

kc = 0.5; %(V/cm)

g = 981; %(cm/s^2)

gamma1 = 0.7; gamma2 = 0.6; %constants that are determined from valve postion

k1 = 3.33; k2 = 3.35; %(cm^3/Vs)

v1 = 3;v2 = 3; %(V)

h0 = [12.4; 12.7; 1.8; 1.4];

C = [kc, 0, 0, 0; 0, kc, 0, 0];

Q = 0.00001*eye(4); %covariance matrix of noise in state equation

R = 0.00001*eye(2); %covariance matrix of noise in measurement equation

%Ordinary Differential Equation System

tspan = linspace(0,6000,10000);

[t,h] = ode45(@ODE,tspan,h0);

y = C*h' + R*randn(2,10000);

%Visualization of Simulation Results

figure;

plot(t,h(:,1),t,h(:,2),t,h(:,3),t,h(:,4))

legend('h1','h2','h3','h4')

title('Simulation results')
```

```
%Initialization

x0 = h0;

Ts = tspan(2)-tspan(1);

n = 4; %Number of states

P0 = 0.0001*eye(4);%Initial Covariance Matrix

N = 400; %Number of particles

L = chol(P0);

x0 = (x0*ones(1,N)) + L*rand(n,N); %Adding covariance to each particle

x_post = x0;

%Roughing the posterior with Noise

w = chol(Q)*rand(n,N);

w1 = w(1,:);

w2 = w(2,:);

w3 = w(3,:);

w4 = w(4,:);

x_post(1,:) = x_post(1,:) + w1;

x_post(2,:) = x_post(2,:) + w2;

x_post(3,:) = x_post(3,:) + w3;

x_post(4,:) = x_post(4,:) + w4;

x_prior_mean = zeros(4,10001);

x_post_mean = zeros(4,10001);

k=1;

while(k<=10000)

w = chol(Q)*randn(n,N); %for roughening the prior

x_dt = zeros(4,N);

%Prediction
```

```
x_dt(1,:) = (-a1/A1*sqrt(2*g*(x_post(1,:))) + a3/A1*sqrt(2*g*(x_post(3,:))) +
(gamma1*k1*v1)/A1);

x_dt(2,:) = -a2/A2*sqrt(2*g*(x_post (2,:))) + a4/A2*sqrt (2*g*(x_post(4,:))) +
(gamma2*k1*v2)/A2 ;

x_dt(3,:) = -a3/A3*sqrt (2*g*(x_post(3,:))) + (1 - gamma2)*k2*v2/A3;

x_dt(4,:) = -a4/A4*sqrt (2*g*(x_post(4,:))) + (1 - gamma1)*k1*v1/A4;

x_prior = x_post + x_dt*Ts + w;

%Importance weights

z_true = y(:,k) * ones(1,N);

z_est = C*x_prior;

v = z_true-z_est;

q = zeros(1,N);

wt = zeros(1,N);

for i=1:N

q(i) = exp(-0.5 * v(:,i)'*inv(R)*v(:,i));

end

for i=1:N

wt(i) = q(i)/sum(q);

end

j=1;

cumsum_wt = cumsum(wt);

for i=1:N

r = rand;

while cumsum_wt(j) < r

j=j+1;

end

x_post(:,i) = x_prior(:,j);
```

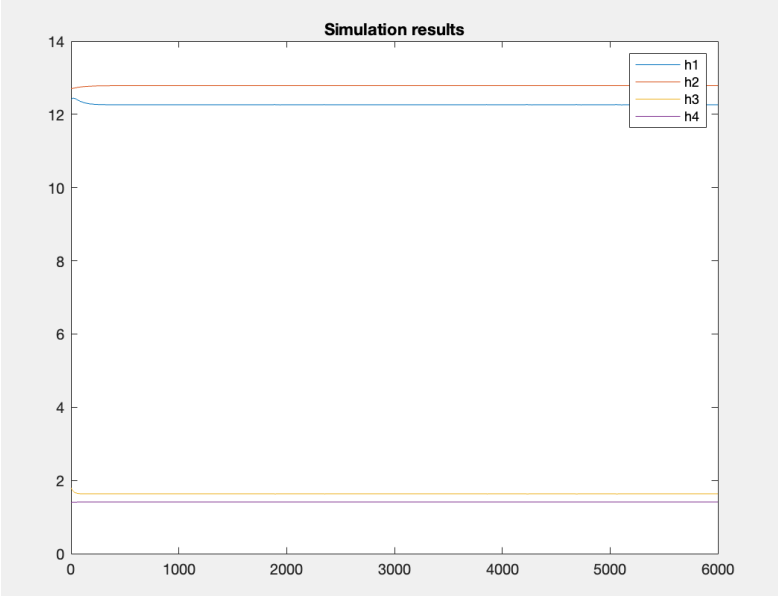
```
end

x_prior_mean(:,k) = mean(x_prior,2);

x_post_mean(:,k) = mean(x_post,2);

k=k+1;

end

%Visualization

k_span = 1:10000;

figure;

plot(k_span,x_post_mean(1,1:10000),k_span,x_post_mean(2,1:10000),k_span,x_post_
mean(3,1:10000),k_span,x_post_mean(4,1:10000));

legend('h1','h2','h3','h4')

title('Particle Filter results')

estm_error = h' - x_post_mean(:,1:10000);

figure;

plot(k_span,estm_error(1,1:10000),k_span,estm_error(2,1:10000),k_span,estm_erro
r(3,1:10000),k_span,estm_error(4,1:10000));

legend('h1','h2','h3','h4')

title('Residuals plot')

figure;

subplot(2,2,1)

plot(k_span,h(:,1)',k_span,x_post_mean(1,1:10000))

title('h1');

subplot(2,2,2)

plot(k_span,h(:,2)',k_span,x_post_mean(2,1:10000))

title('h2');

subplot(2,2,3)

plot(k_span,h(:,3)',k_span,x_post_mean(3,1:10000))
```

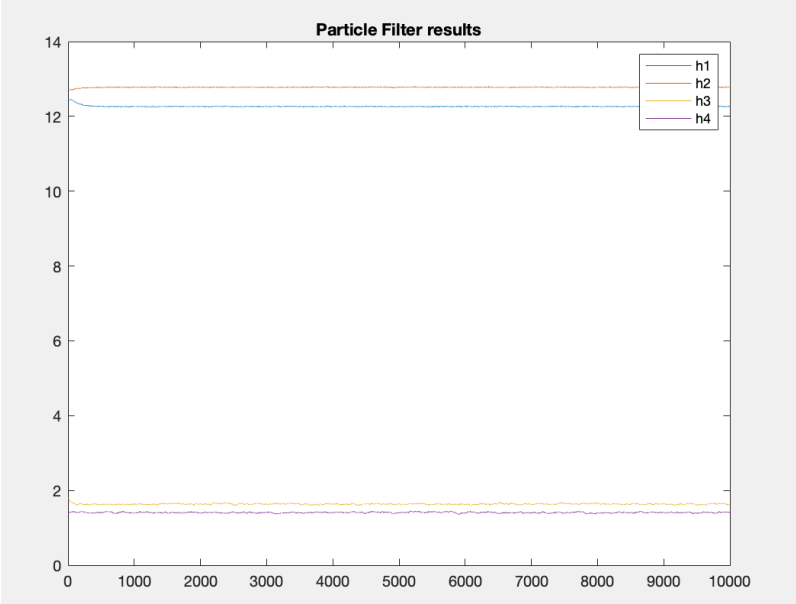
```
title('h3');

subplot(2,2,4)

plot(k_span,h(:,4)',k_span,x_post_mean(4,1:10000))

title('h4');

sgtitle('Comparing ')

function dhdt = ODE(t,h)

A1 = 28; %(cm^2)

A2 = 32;

A3 = 28;

A4 = 32;

a1 = 0.071; a3 = 0.071; %(cm^2)

a2 = 0.057; a4 = 0.057;

kc = 0.5; %(V/cm)

g = 981; %(cm/s^2)

gamma1 = 0.7; gamma2 = 0.6; %constants that are determined from valve postion

k1 = 3.33; k2 = 3.35; %(cm^3/Vs)

v1 = 3;v2 = 3; %(V)

dhdt = zeros(4,1);

dhdt(1) = -a1/A1*(2*g*h(1))^0.5 + a3/A1*(2*g*h(3))^0.5 + gamma1*k1/A1*v1;

dhdt(2) = -a2/A2*(2*g*h(2))^0.5 + a4/A2*(2*g*h(4))^0.5 + gamma2*k2/A2*v2;

dhdt(3) = -a3/A3*(2*g*h(3))^0.5 + (1-gamma2)*k2*v2/A3;

dhdt(4) = -a4/A4*(2*g*h(4))^0.5 + (1-gamma1)*k1*v1/A4;

end
```

Assumptions:

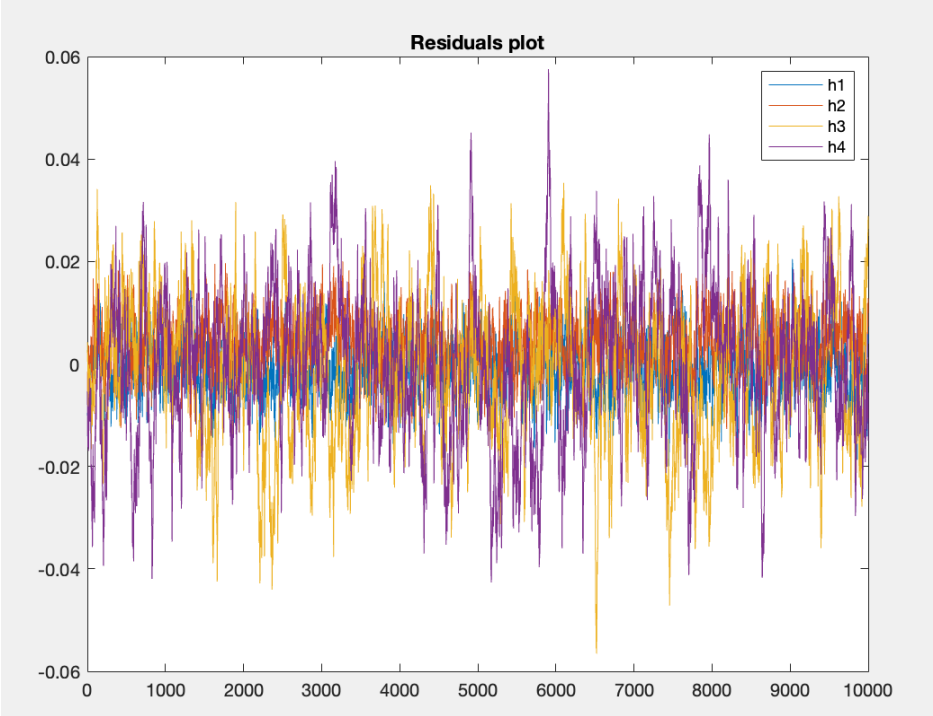
- **State Space Representation:** Particle filters assume that the system can be described by a state-space model.
- **Noisy Measurements:** Particle filters typically assume that the measurements or observations of the system are noisy. The noise in measurements is often modeled as Gaussian, but it doesn't have to be. Particle filters can handle non-Gaussian noise.
- **Independence Assumption:** Particles in a particle filter are assumed to be independent and identically distributed (i.i.d.). Each particle represents a possible state hypothesis, and their independence is essential for the resampling step, where particles with higher weights are replicated.
- **Non-Linearity and Non-Gaussianity:** Particle filters are particularly well-suited for non-linear and non-Gaussian state estimation problems. They do not assume that the system dynamics or the state and measurement models follow linear or Gaussian distributions.

Plots:-

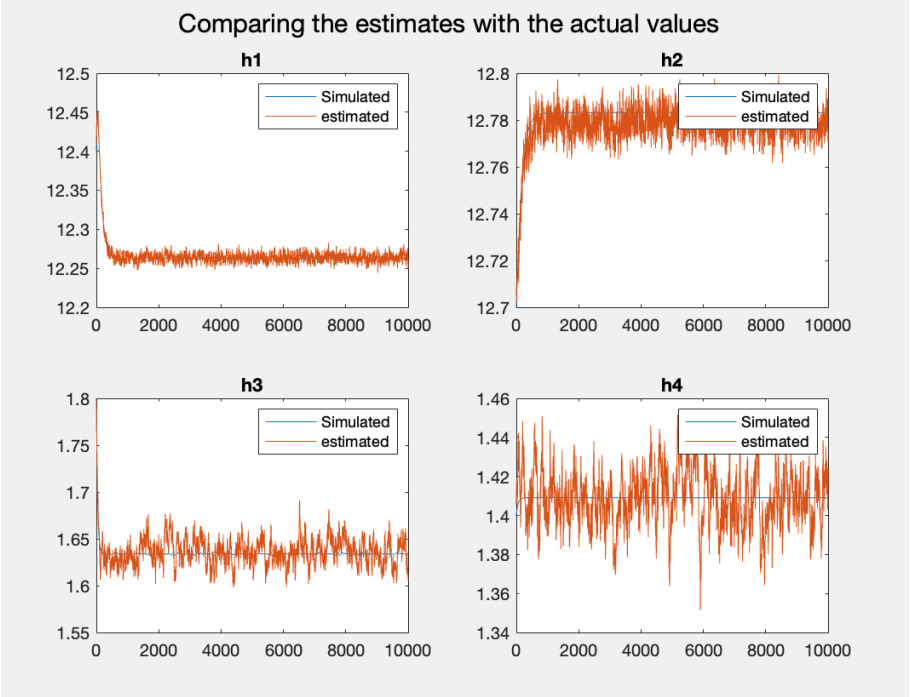




The first plot represents the results of the simulation, and the second plot represents the estimations made by the particle filter. At first glance, both the plots look almost identical. This fact is further corroborated by the following residual plot.



The residuals here represent the difference between the estimation and simulation. From the plot, we can see that the maximum magnitude of the residual is near 0.06. This extremely small value confirms that the particle filter has done an impressive job at estimating the states of the system.



Final Inferences:-

In case of particle filters, the optimization was primarily concerned with ensuring that the particle filter converges. Due to the randomness associated with each particle, the number of iterations until convergence varies each time the code is run. Therefore a valid comparison between different values of Q and R is not possible. After several tries, Q = 0.00001 and R = 0.00001 were found to allow for filter convergence and thus, these values were chosen.

The particle filter was successfully implemented using 500 particles to estimate and track the states of the system.