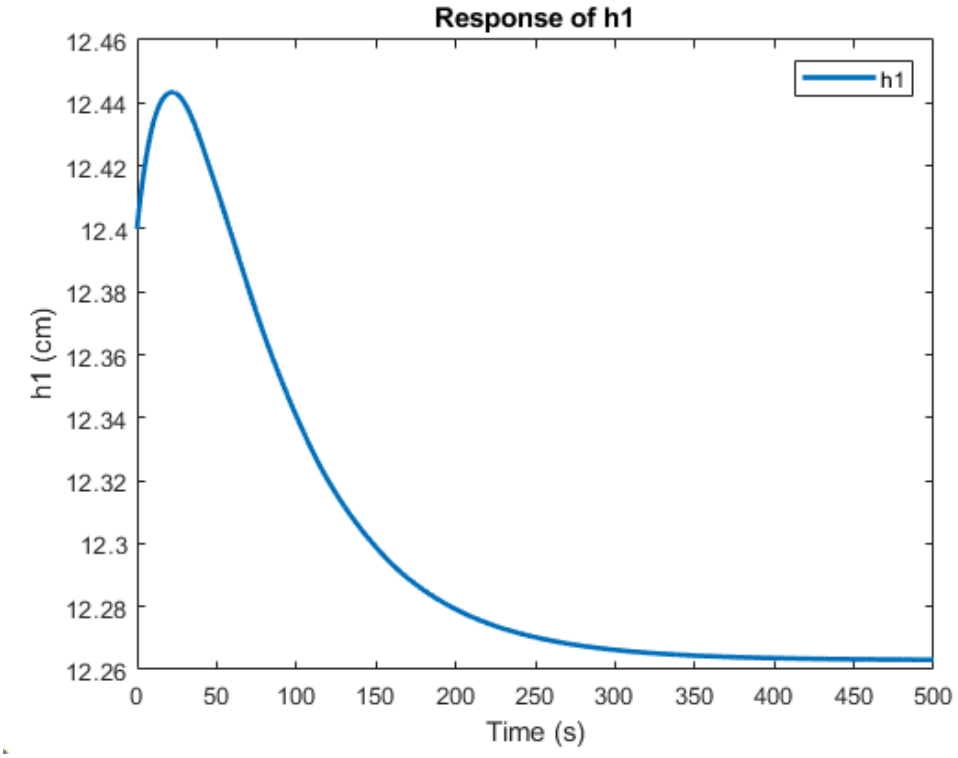
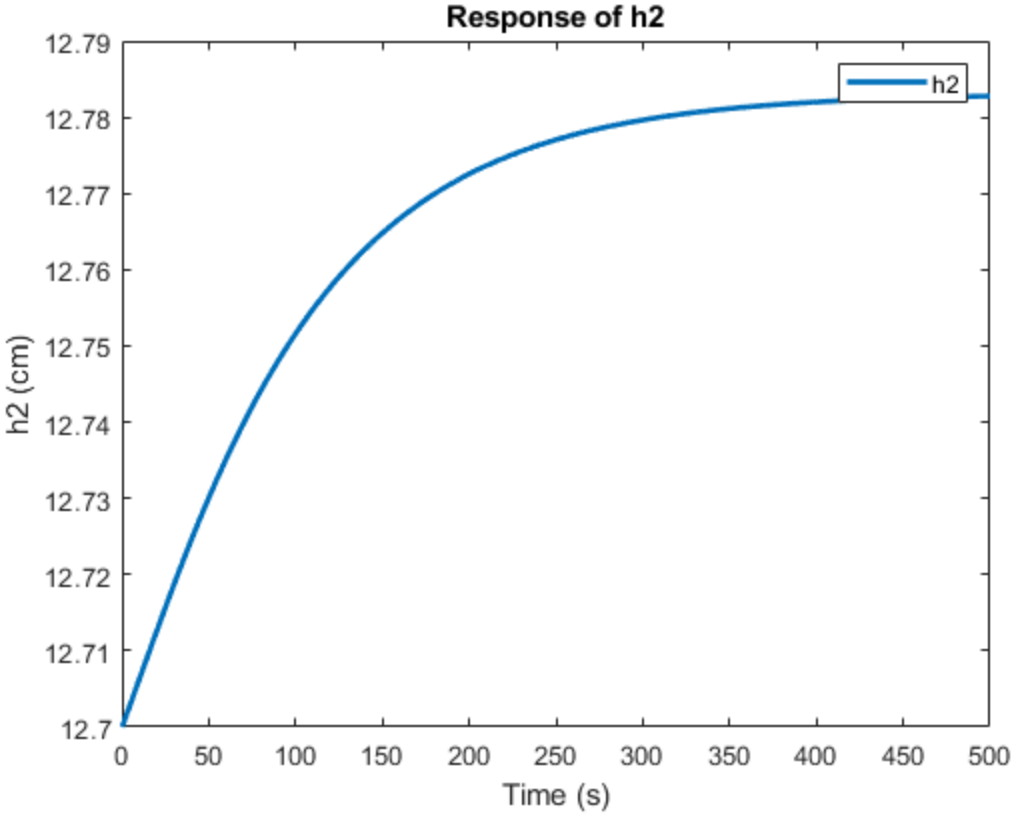
Solution 1

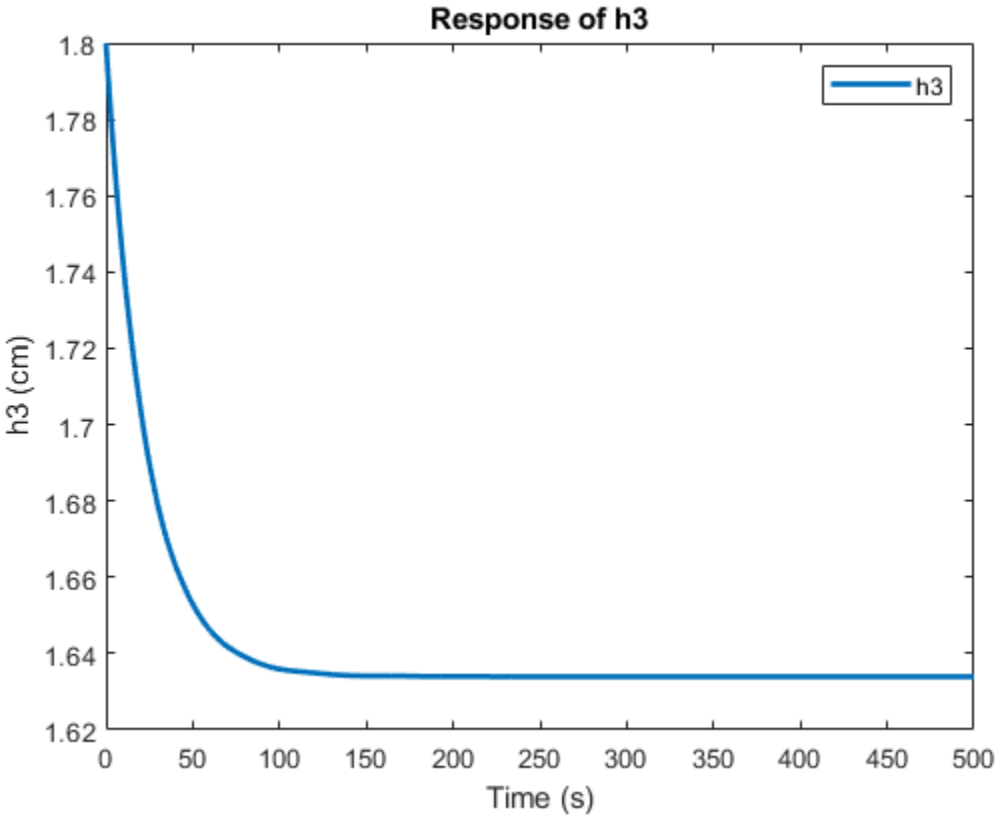
Solving the set of nonlinear equations with the help of ode45 solved in matlab and plotting h1, h2, h3 and h4. This data is further used to compare data obtained from kalman and particle filter.

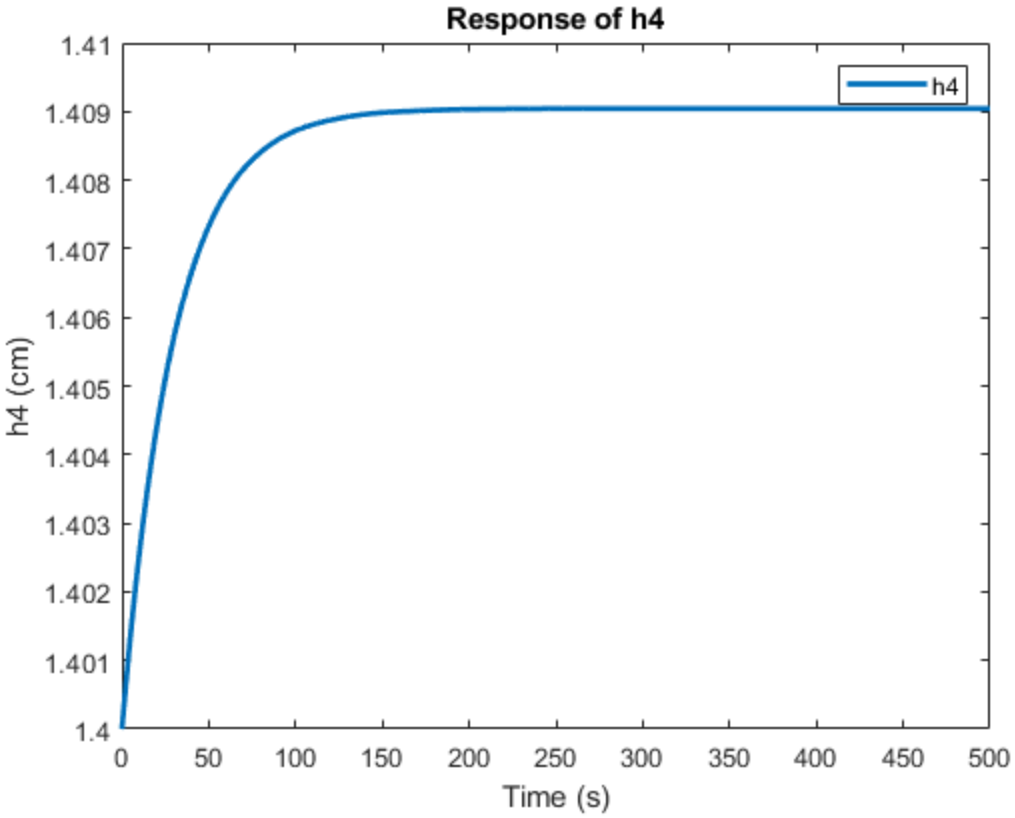
```
% function four_tank_system
clear all;
clc;
% Initial conditions
h0 = [12.4; 12.7; 1.8; 1.4];  % Initial water levels in the tanks (cm)
% Time span for the solution
tspan = [0, 500]; % Set the final time t_final
% Solve the system using ODE45
[t, h] = ode45(@four_tank_eqns, linspace(tspan(1), tspan(2), 1000), h0);

subplot(2, 2, 1);
plot(t,h(:,1))
xlabel('Time (s)');
ylabel('h1 (cm)');
legend('h1');
subplot(2, 2, 2);
plot(t,h(:,2))
xlabel('Time (s)');
ylabel('h2 (cm)');
legend('h2');
subplot(2, 2, 3);
plot(t,h(:,3))
xlabel('Time (s)');
ylabel('h3 (cm)');
legend('h3');
subplot(2, 2, 4);
plot(t,h(:,4))
xlabel('Time (s)');
ylabel('h4 (cm)');
legend('h4');
title('Four-Tank System Response');
% Function that defines the system of equations
function dhdt = four_tank_eqns(t, h)
% Extract variables
h1 = h(1);
h2 = h(2);
h3 = h(3);
h4 = h(4);

A=[28;32;28;32];
a=[0.071;0.057;0.071;0.057];
kc = 0.5;  % Sensor gain (V/cm)
```

```
    g = 981;  % Gravitational acceleration (cm/s^2)
    % Valve parameters
    gamma1 = 0.7; gamma2 = 0.6;  % Constants determined by valve positions
    % Pump parameters
    k1 = 3.33; k2 = 3.35;  % Pump constants (cm^3/V/s)
    v1 = 3; v2 = 3;  % Voltage input to the pumps (V)
    % Define the differential equations
    dh1dt = - (a(1)/A(1)) * sqrt(2*g*h1) + (a(3)/A(1)) * sqrt(2*g*h3) +
(gamma1*k1/A(1)) * v1;
    dh2dt = - (a(2)/A(2)) * sqrt(2*g*h2) + (a(4)/A(2)) * sqrt(2*g*h4) +
(gamma2*k2/A(2)) * v2;
    dh3dt = - (a(3)/A(3)) * sqrt(2*g*h3) + ((1 - gamma2)*k2/A(3)) * v2;
    dh4dt = - (a(4)/A(4)) * sqrt(2*g*h4) + ((1 - gamma1)*k1/A(4)) * v1;
    % Output
    dhdt = [dh1dt; dh2dt; dh3dt; dh4dt];
end
```




Response of h2
h2
h2 (cm)
Time (s)
12.79
12.78
12.77
12.76
12.75
12.74
12.73
12.72
12.71
12.7
0
50
100
150
200
250
300
350
400
450
500


Response of h3
h3
h3 (cm)
Time (s)
1.8
1.78
1.76
1.74
1.72
1.7
1.68
1.66
1.64
1.62
0
50
100
150
200
250
300
350
400
450
500



As it can be seen after using ode45 solver for nonlinear equations, h1, h2, h3 and h4 settles for values **12.27cm, 12.78cm, 1.64cm, 1.409cm** respectively.

PART A

Solution 2

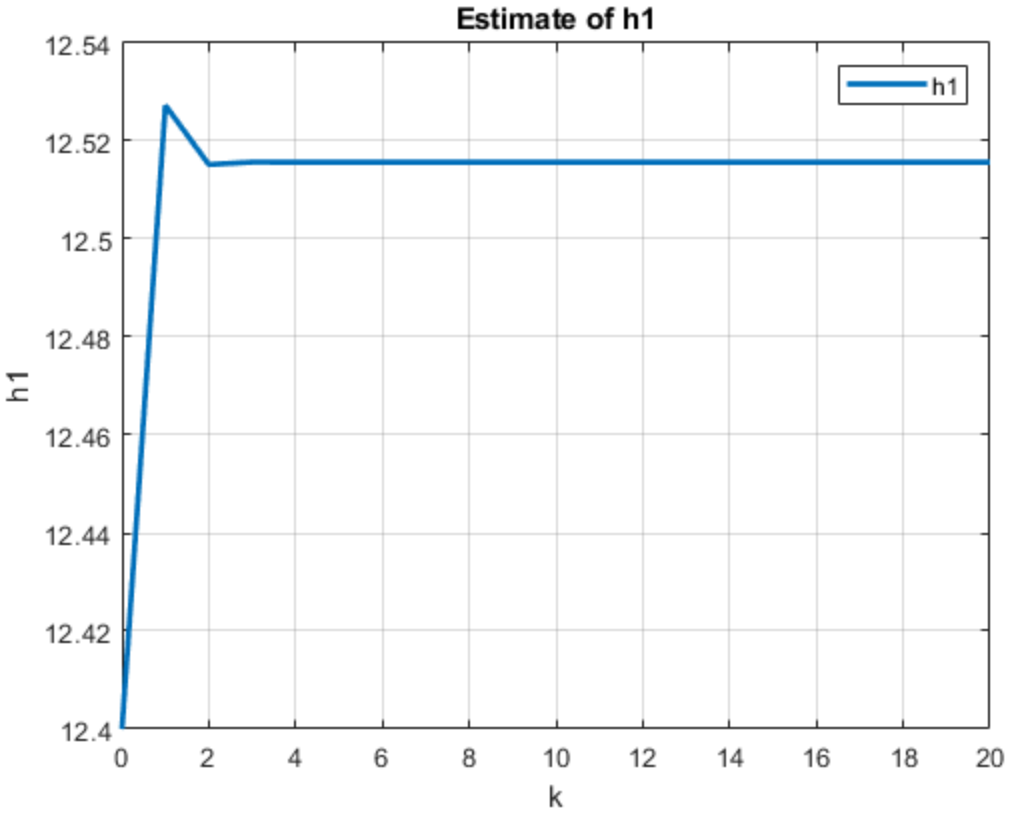
Code for kalman filter is given below:

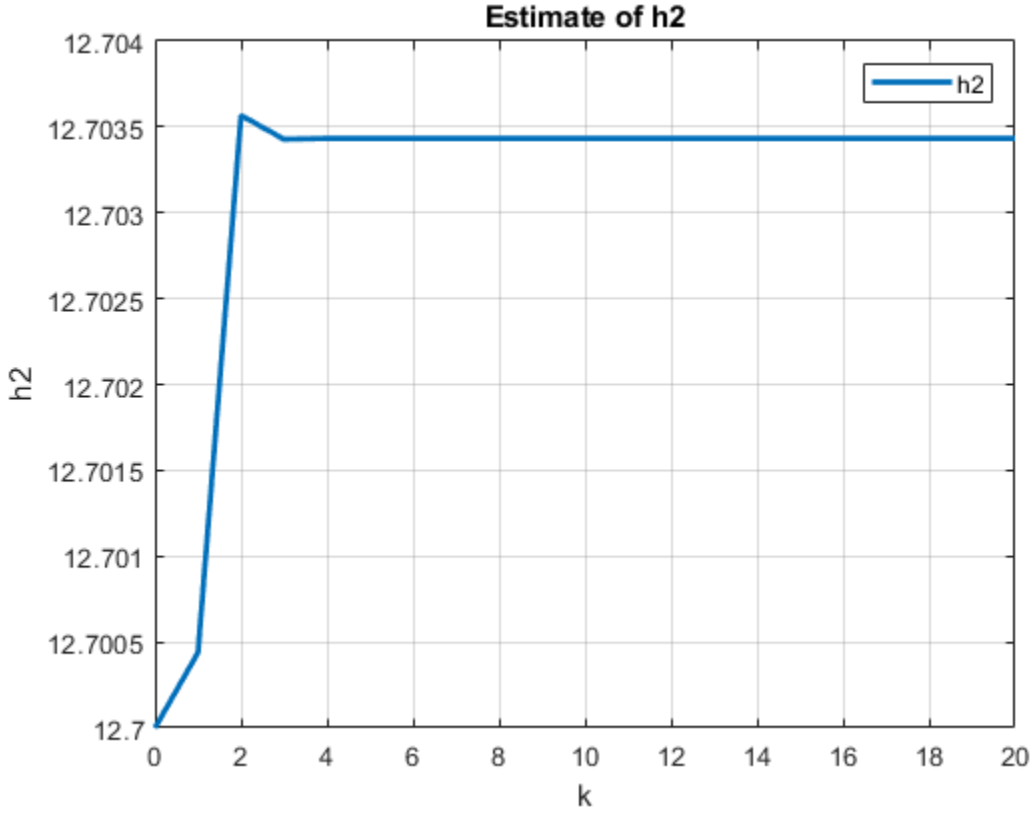
```
clear all;
Clc;

% Tank and system parameters
A = [28; 32; 28; 32];  % Cross-sectional areas of the tanks (cm^2)
a = [0.071; 0.057; 0.071; 0.057];  % Outlet areas (cm^2)
kc = 0.5;  % Sensor gain (V/cm)
g = 981;  % Gravitational acceleration (cm/s^2)
% Valve parameters
gamma1 = 0.7; gamma2 = 0.6;  % Constants determined by valve positions
% Pump parameters
k1 = 3.33; k2 = 3.35;  % Pump constants (cm^3/V/s)
v1 = 3; v2 = 3;  % Voltage input to the pumps (V)
% Initial conditions
h0 = [12.4; 12.7; 1.8; 1.4];  % Initial water levels in the tanks (cm)
% A, B, and C matrices

for i = 1:4
   T(i) = (A(i) / a(i)) * (sqrt(2 * h0(i) / g));
End

Ad = [-1/T(1) 0 A(3)/(A(1)*T(3)) 0;
     0 -1/T(2) 0 A(4)/(A(2)*T(4));
     0 0 -1/T(3) 0;
     0 0 0 -1/T(4)];
Bd = [gamma1 * k1 / A(1) 0;
     0 gamma2 * k2 / A(2);
     0 (1 - gamma2) * k2 / A(3);
     (1 - gamma1) * k1 / A(4) 0];
H = [kc 0 0 0;
    0 kc 0 0];
U = [v1; v2];

X_post = h0;  % Initial state estimate
P_post = 100 * eye(4);  % Initial estimate covariance

% Process and measurement noise covariance matrices
Q = [90, 30, 20, 10;  % Tank 1
    30, 90, 15, 5;   % Tank 2
    0, 15, 50, 0;    % Tank 3
    10, 5, 25, 50];    % Tank 4
R =  30* eye(2);
```

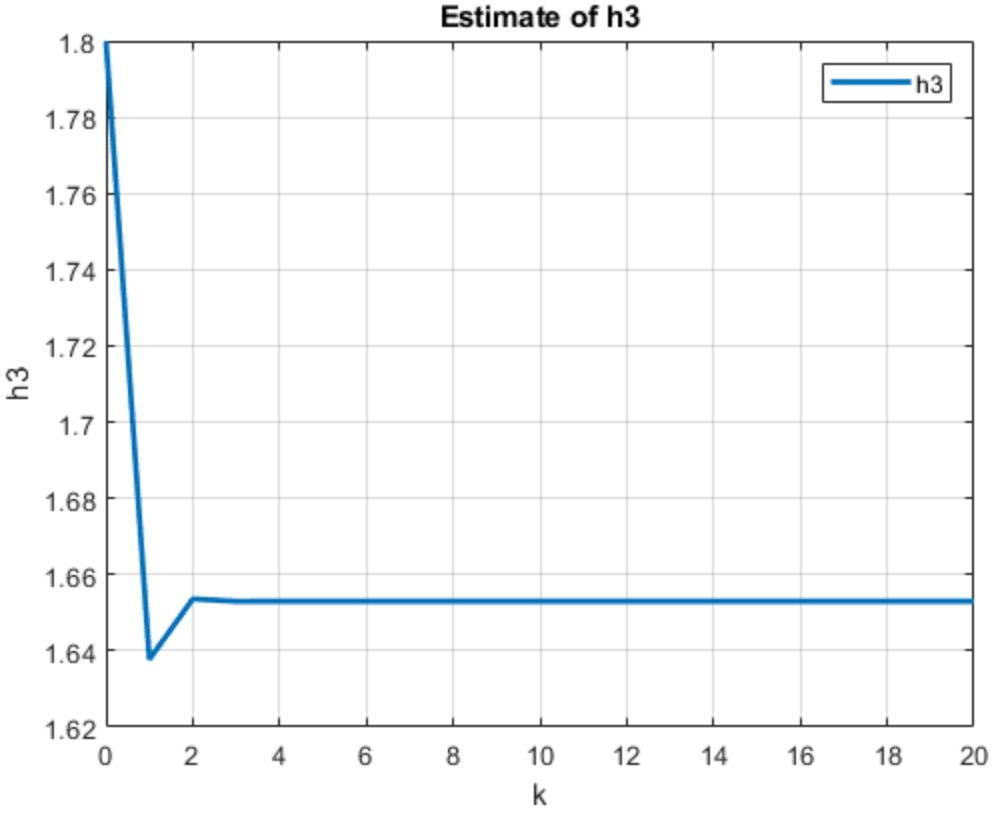
```
% Measurement vector (true measurements of h1 and h2)
Z_true = [12.4; 12.7];

% Number of iterations
num_iterations = 20;
X_estimated = zeros(4, num_iterations);
X_estimated(:, 1) = X_post;
X_prior_estimated=zeros(4, num_iterations);
P_estimated = zeros(4,4, num_iterations);
P_estimated(:,:, 1) = P_post;
P_prior_estimated=zeros(4,4, num_iterations);

for k = 1:num_iterations
   % Prediction step
   X_prior = Ad * X_post + Bd * U;
   P_prior = Ad * P_post * Ad' + Q;

   % Measurement update step
   K = P_prior * H' / (H * P_prior * H' + R);
   X_post = X_prior + K * (Z_true - H * X_prior);
   P_post = (eye(4) - K * H) * P_prior;

   % Store the results
   X_estimated(:, k+1) = X_post;
   P_estimated(:,:, k+1) = P_post;
   X_prior_estimated(:, k+1)= X_prior;
   P_prior_estimated(:,:,k+1)=P_prior;
end
k = 0:num_iterations;
% Figure for h1 (first row of X_estimated)
figure;
plot(k, X_estimated(1, :), 'LineWidth', 2);
title('Estimate of h1');
xlabel('k');
ylabel('h1');
legend('h1');
grid on;
% Figure for h2 (second row of X_estimated)
figure;
plot(k, X_estimated(2, :), 'LineWidth', 2);
title('Estimate of h2');
xlabel('k');
ylabel('h2');
legend('h2');
grid on;
% Figure for h3 (third row of X_estimated)
```

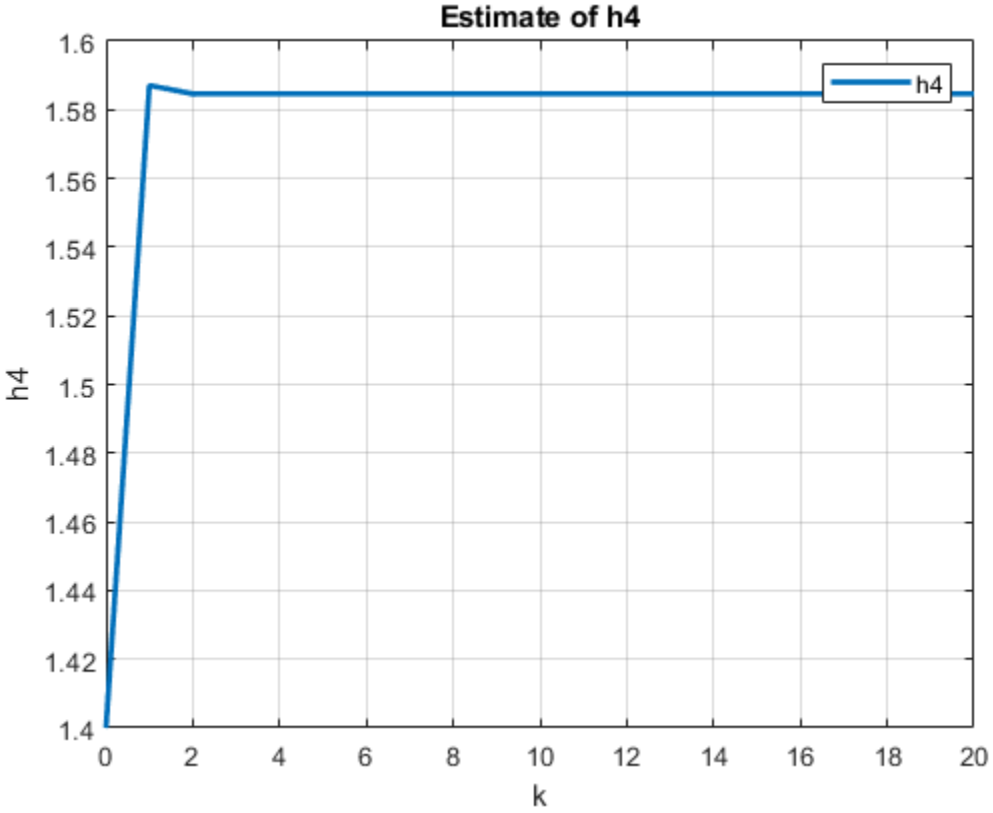
```
figure;
plot(k, X_estimated(3, :), 'LineWidth', 2);
title('Estimate of h3');
xlabel('k');
ylabel('h3');
legend('h3');
grid on;
% Figure for h4 (fourth row of X_estimated)
figure;
plot(k, X_estimated(4, :), 'LineWidth', 2);
title('Estimate of h4');
xlabel('k');
ylabel('h4');
legend('h4');
grid on;
```

Results:


Estimate of h1
h1
12.54
12.52
12.5
12.48
12.46
12.44
12.42
12.4
0
2
4
6
8
10
12
14
16
18
20
k


Estimate of h2
h2
12.704
12.7035
12.703
12.7025
12.702
12.7015
12.701
12.7005
12.7
0
2
4
6
8
10
12
14
16
18
20
k


Estimate of h3
h3
k
1.8
1.78
1.76
1.74
1.72
1.7
1.68
1.66
1.64
1.62
0
2
4
6
8
10
12
14
16
18
20


Estimate of h4
h4
k
1.6
1.58
1.56
1.54
1.52
1.5
1.48
1.46
1.44
1.42
1.4
0
2
4
6
8
10
12
14
16
18
20

As it can be seen using the kalman filter, h1, h2, h3 and h4 settle for values 12.515, 12.7034cm,1.65cm, 1.584cm respectively. The estimated values are quite close to values obtained from the ode45 solver which validates our kalman filter.

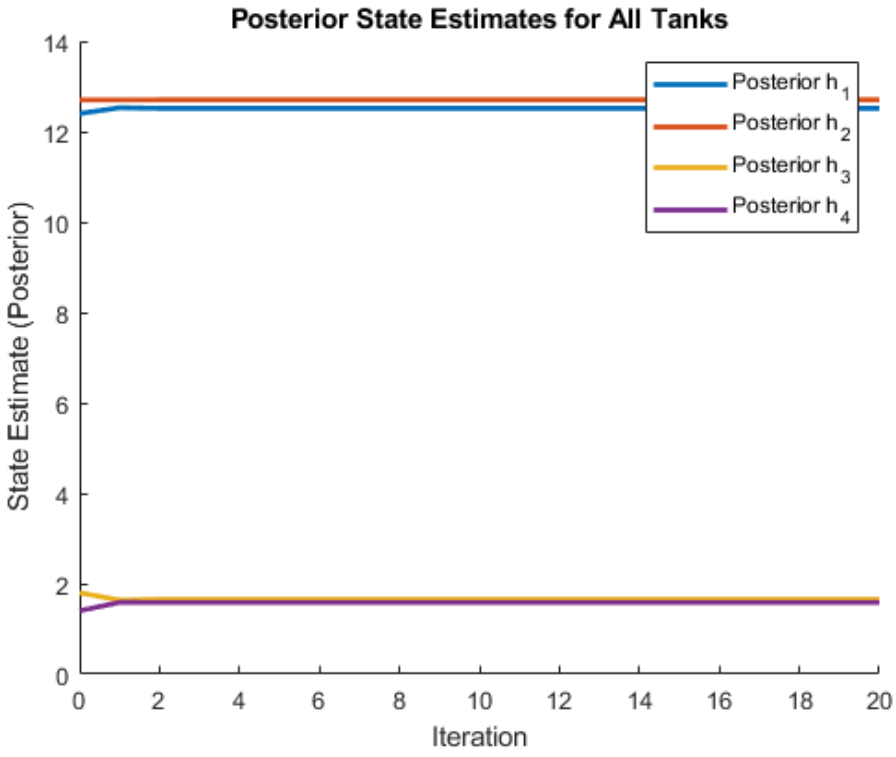
Solution 3

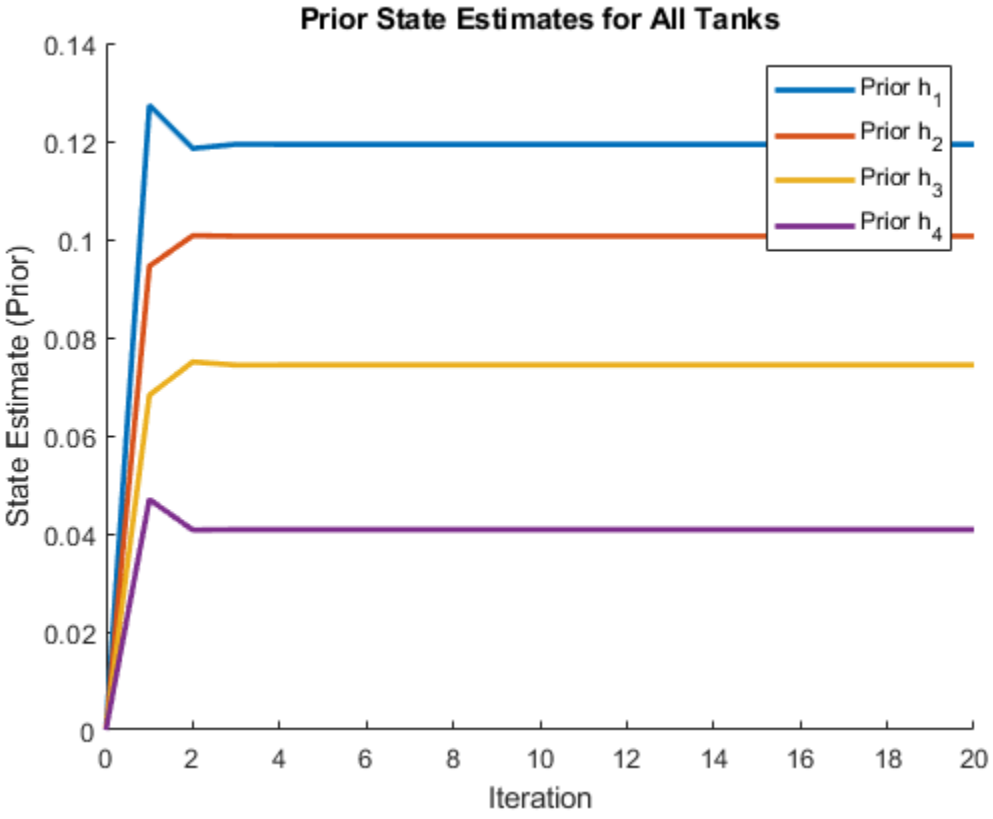
Code for plotting graphs for understanding the performance of kalman filter is given below:

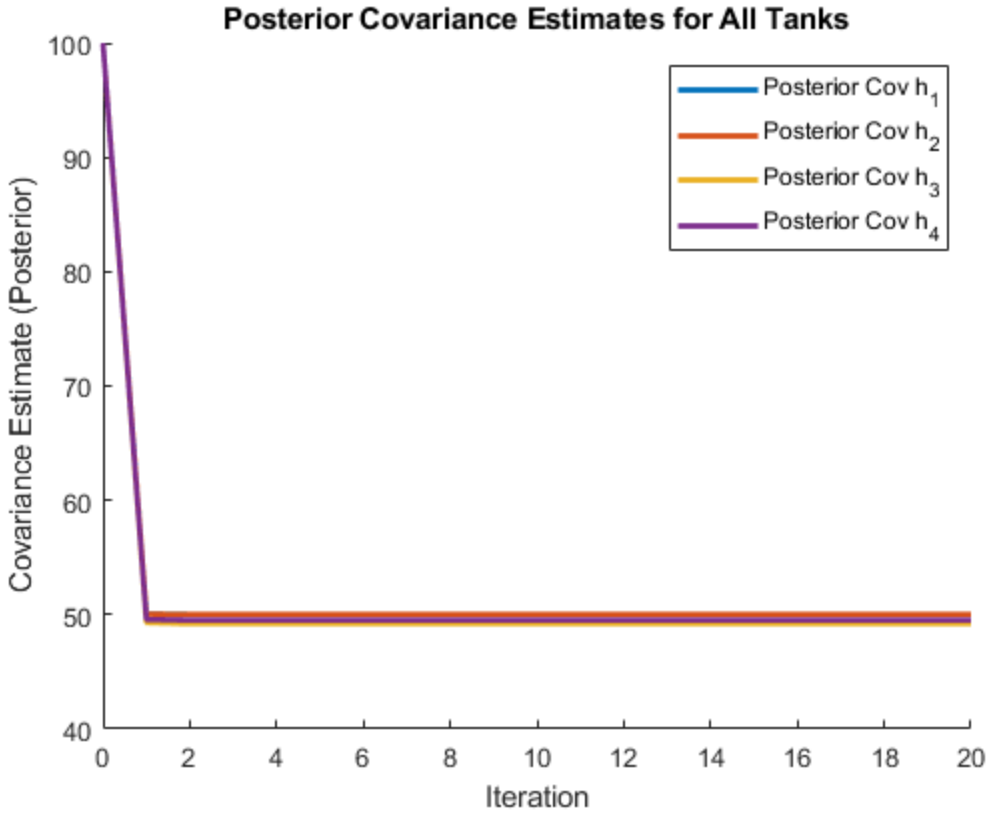
```
k = 0:num_iterations;
% Plotting all Prior State Estimates in one figure
figure;
hold on;
for i = 1:4
   plot(k, X_prior_estimated(i, :), 'DisplayName', ['Prior h_'
num2str(i)],'LineWidth', 2);
end
xlabel('Iteration');
ylabel('State Estimate (Prior)');
legend;
title('Prior State Estimates for All Tanks');
hold off;
% Plotting all Posterior State Estimates in another figure
figure;
hold on;
for i = 1:4
   plot(k, X_estimated(i, :), 'DisplayName', ['Posterior h_'
num2str(i)],'LineWidth', 2);
end
xlabel('Iteration');
ylabel('State Estimate (Posterior)');
legend;
title('Posterior State Estimates for All Tanks');
hold off;
% Plotting all Prior Covariance Estimates in one figure (diagonal elements)
figure;
hold on;
for i = 1:4
   plot(k, squeeze(P_prior_estimated(i, i, :)), 'DisplayName', ['Prior Cov h_'
num2str(i)],'LineWidth', 2);
end
xlabel('Iteration');
ylabel('Covariance Estimate (Prior)');
legend;
title('Prior Covariance Estimates for All Tanks');
hold off;
% Plotting all Posterior Covariance Estimates in another figure (diagonal elements)
```

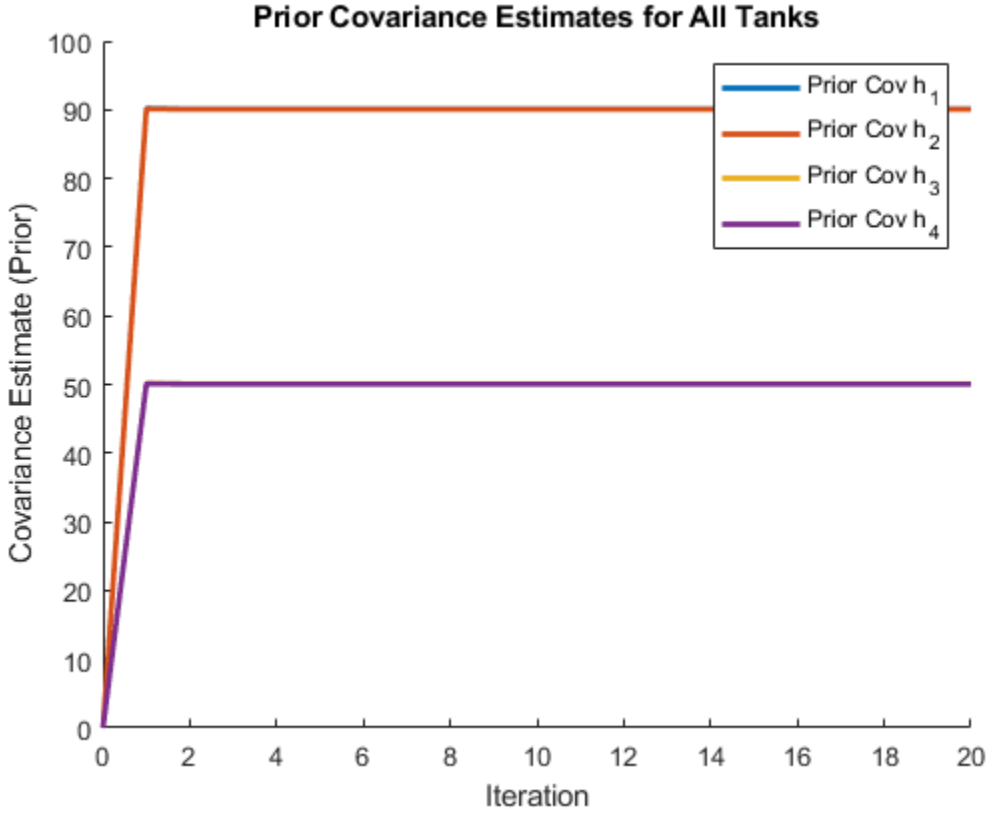
```
figure;
hold on;
for i = 1:4
   plot(k, squeeze(P_estimated(i, i, :)), 'DisplayName', ['Posterior Cov h_'
num2str(i)],'LineWidth', 2);
end
xlabel('Iteration');
ylabel('Covariance Estimate (Posterior)');
legend;
title('Posterior Covariance Estimates for All Tanks');
hold off;
% Plotting Kalman Gain (K(k))
figure;
for i = 1:4
   for j = 1:2
       subplot(4, 2, (i-1)*2 + j);
       plot(k, squeeze(K_storage(i, j, :)), 'r', 'DisplayName', ['K' num2str(i)
num2str(j)], 'LineWidth', 2);
       xlabel('Iteration');
       ylabel(['K' num2str(i) num2str(j)]);
       legend;
       title(['Kalman Gain K' num2str(i) num2str(j)]);
   end
end
% Plotting Prior and Posterior Residues/Innovations (r(k-) and r(k+))
prior_residues = Z_true - H * X_prior_estimated;
posterior_residues = Z_true - H * X_estimated;
% Plotting prior residues in one figure
figure;
hold on;
plot(k, prior_residues(1, :), 'b', 'DisplayName', 'Prior r_1', 'LineWidth', 2);
plot(k, prior_residues(2, :), 'b--', 'DisplayName', 'Prior r_2', 'LineWidth', 2);
xlabel('Iteration');
ylabel('Prior Residues');
legend;
title('Prior Residues for r_1 and r_2');
% Plotting posterior residues in another figure
figure;
hold on;
plot(k, posterior_residues(1, :), 'r', 'DisplayName', 'Posterior r_1', 'LineWidth',
2);
plot(k, posterior_residues(2, :), 'r--', 'DisplayName', 'Posterior r_2',
'LineWidth', 2);
xlabel('Iteration');
ylabel('Posterior Residues');
legend;
title('Posterior Residues for r_1 and r_2');
```

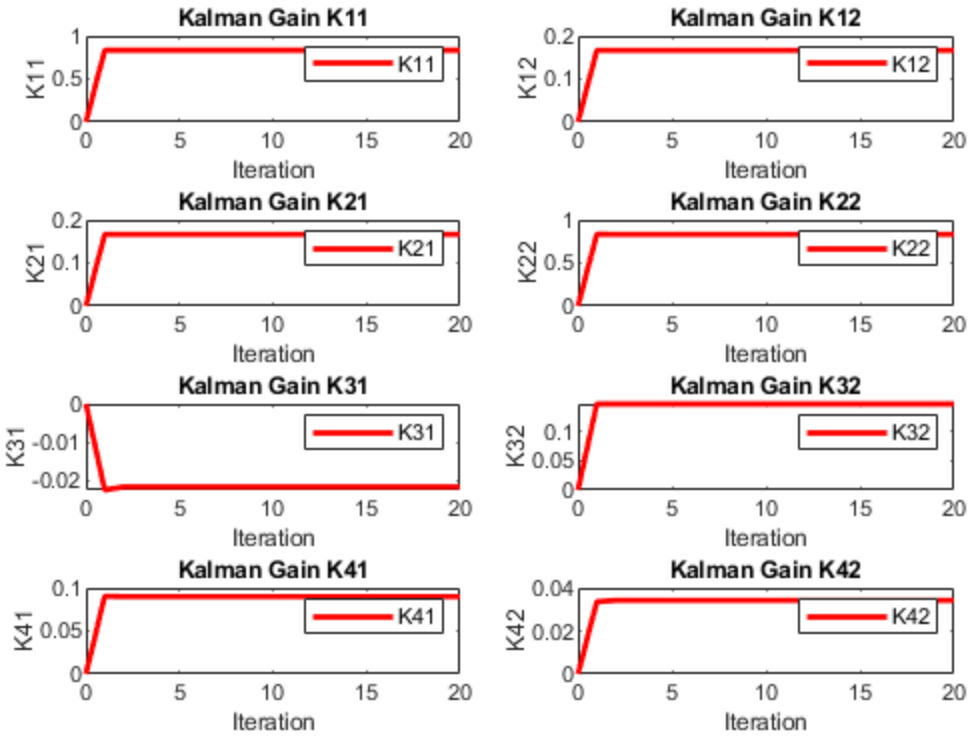
Resulted plots:

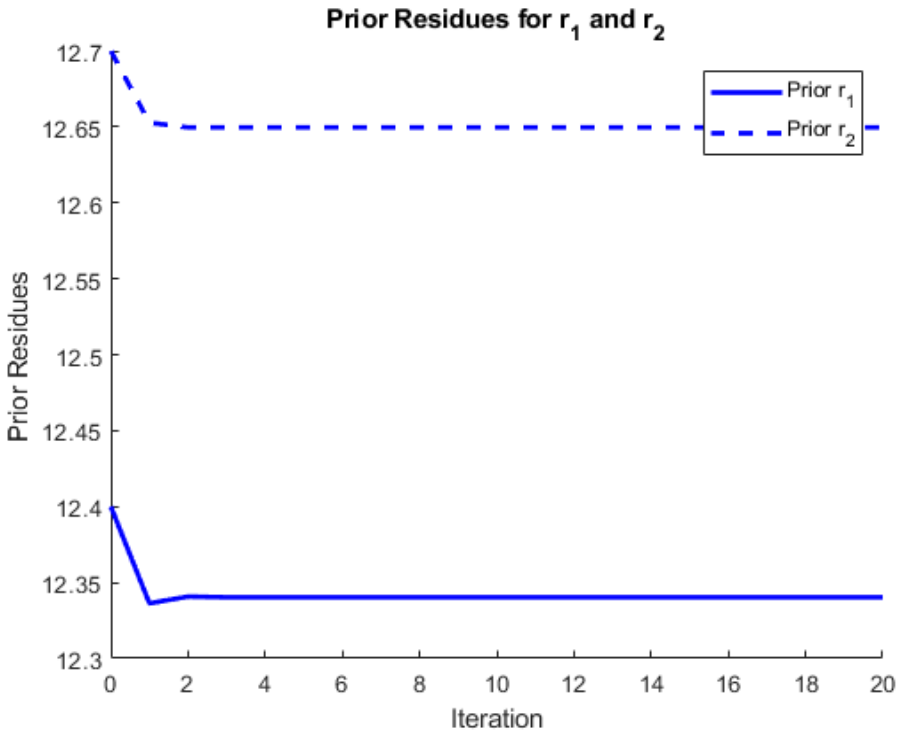


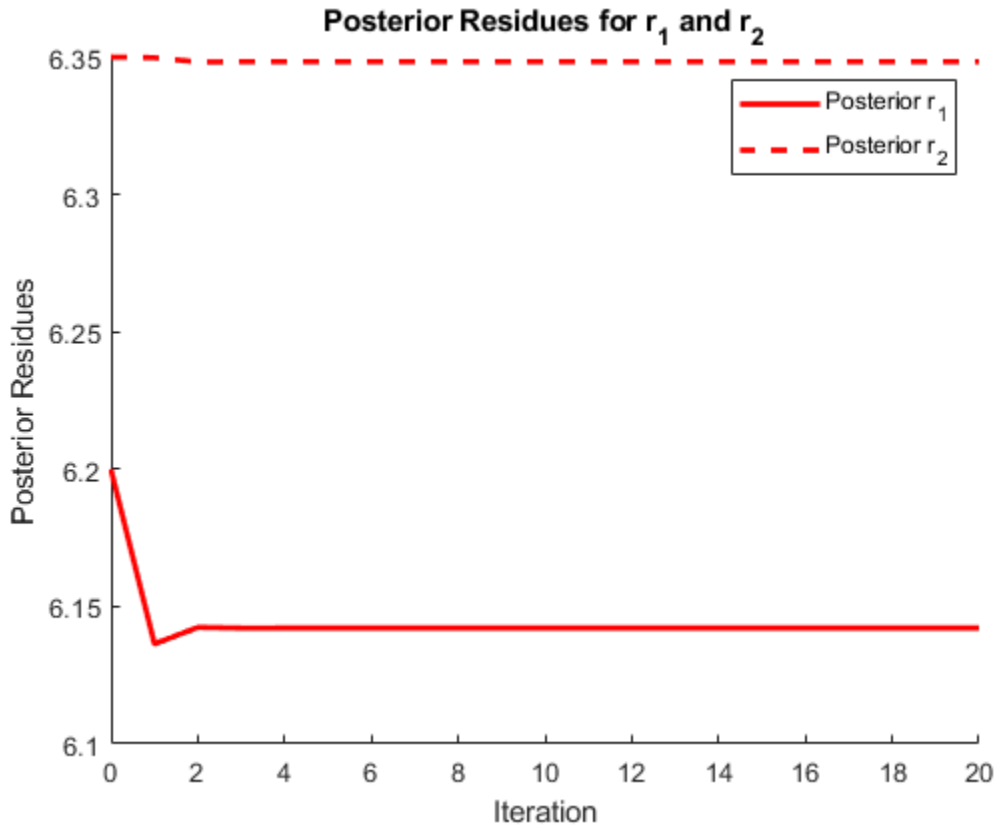



Posterior Covariance Estimates for All Tanks
Posterior Cov h1
Posterior Cov h2
Posterior Cov h3
Posterior Cov h4
Covariance Estimate (Posterior)
Iteration
100
90
80
70
60
50
40
0
2
4
6
8
10
12
14
16
18
20


Prior Covariance Estimates for All Tanks
Prior Cov h1
Prior Cov h2
Prior Cov h3
Prior Cov h4
Covariance Estimate (Prior)
Iteration
100
90
80
70
60
50
40
30
20
10
0
0
2
4
6
8
10
12
14
16
18
20


Kalman Gain K11
K11
Iteration
Kalman Gain K12
K12
Iteration
Kalman Gain K21
K21
Iteration
Kalman Gain K22
K22
Iteration
Kalman Gain K31
K31
Iteration
Kalman Gain K32
K32
Iteration
Kalman Gain K41
K41
Iteration
Kalman Gain K42
K42
Iteration


Prior Residues for r1 and r2
Prior r1
Prior r2
Prior Residues
Iteration
12.7
12.65
12.6
12.55
12.5
12.45
12.4
12.35
12.3
0
2
4
6
8
10
12
14
16
18
20


Posterior Residues for r1 and r2
Posterior r1
Posterior r2
Posterior Residues
Iteration
6.35
6.3
6.25
6.2
6.15
6.1
0
2
4
6
8
10
12
14
16
18
20