Nilesh Balu
ME20B121

CH5120 MODERN CONTROL THEORY
MINI PROJECT 1

Problem Statement:
To estimate the level of water in the four tanks present in the Quadruple tank experiment using the following methods:

1. Kalman filter
2. Particle filter

Part 1: The Kalman Filter:
The model of the quadruple tank problem is given to us in the form of A, B, C and D matrices. Using this information, I implemented the Kalman filter and analysed the result.

The code:
Link: Kalman Filter

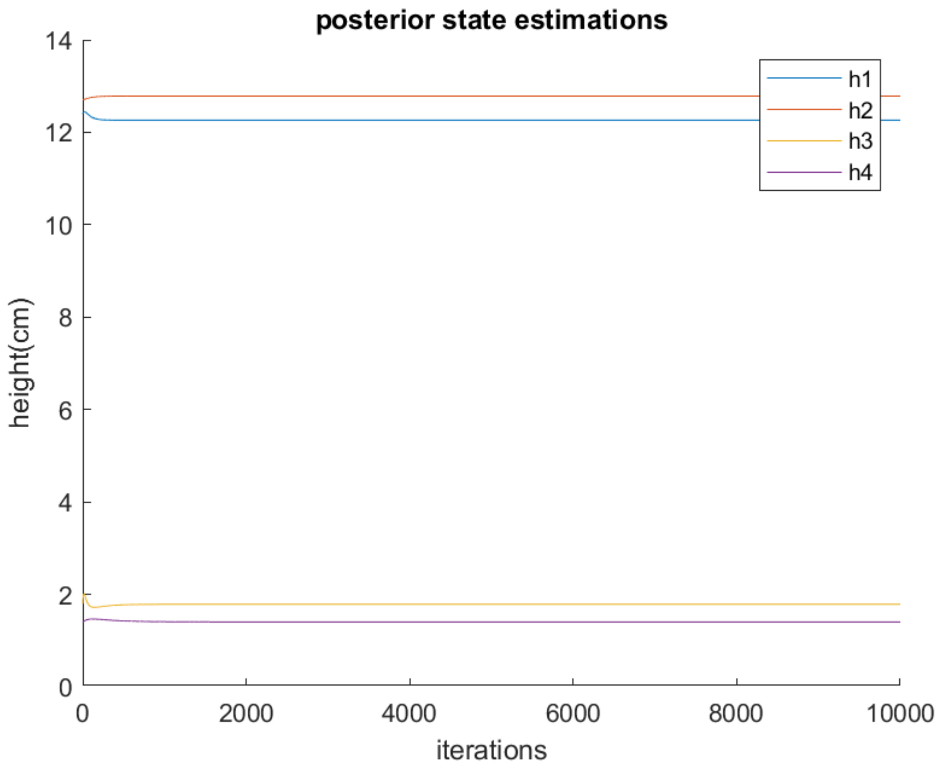
```
%predict step
x_prior = A*x_posterior + B*u;
P_prior = A*P_posterior*A' + Q;
y_prior = C*x_prior;
%finding the kalman gain
innovation = [y(i,1:2)]' - y_prior;
K = P_prior*C'*inv(C*P_prior*C' + R);
%update step
x_posterior = x_prior + K*innovation;
P_posterior = P_prior - K*C*P_prior;
y_posterior = C*x_posterior;
residue = [y(i,1:2)]' - y_posterior;
%storing data to plot
P_prior_plot(i) = norm(P_prior);
P_posterior_plot(i) = norm(P_posterior);
K_plot(i,:,:) = K;
residue_plot(i,:) = residue;
innovation_plot(i,:) = innovation;
h_posterior(i,:) = [12.4;12.7;1.8;1.4] + x_posterior;
h_prior(i,:) = [12.4;12.7;1.8;1.4] + x_prior;
%keyboard
```

Results:

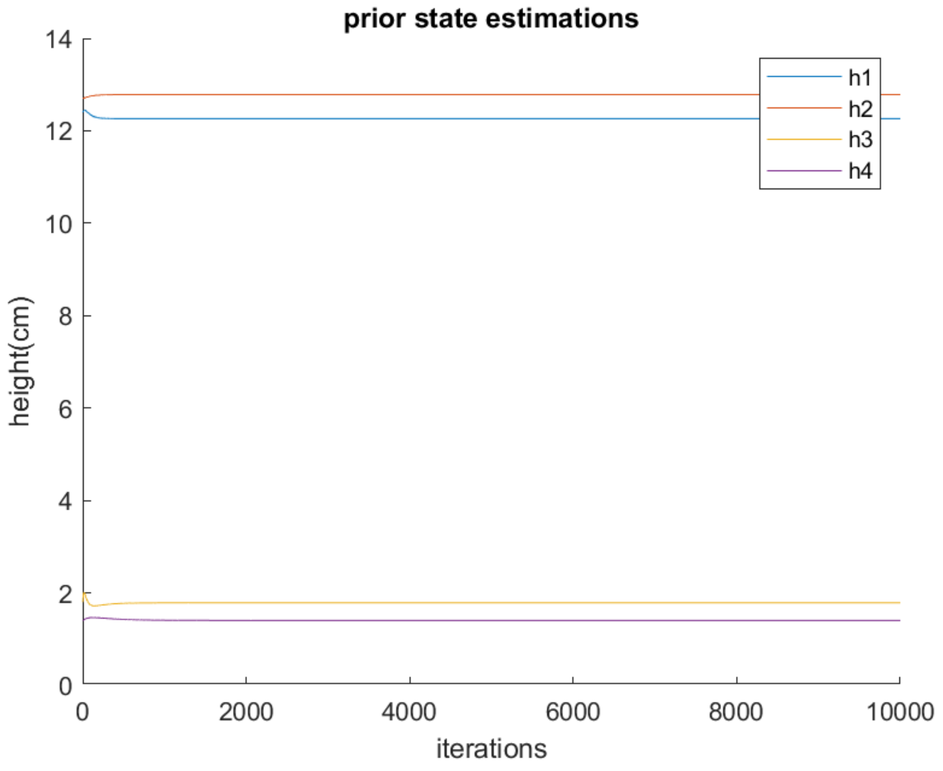
```
tolerance =

  1.8463e-04


h =

  12.2630
  12.7832
   1.7851
   1.4043
```

Here, tolerance represents the L2 norm between the prior and posterior state vectors, and h represents the final estimates of the water levels of the 4 tanks.

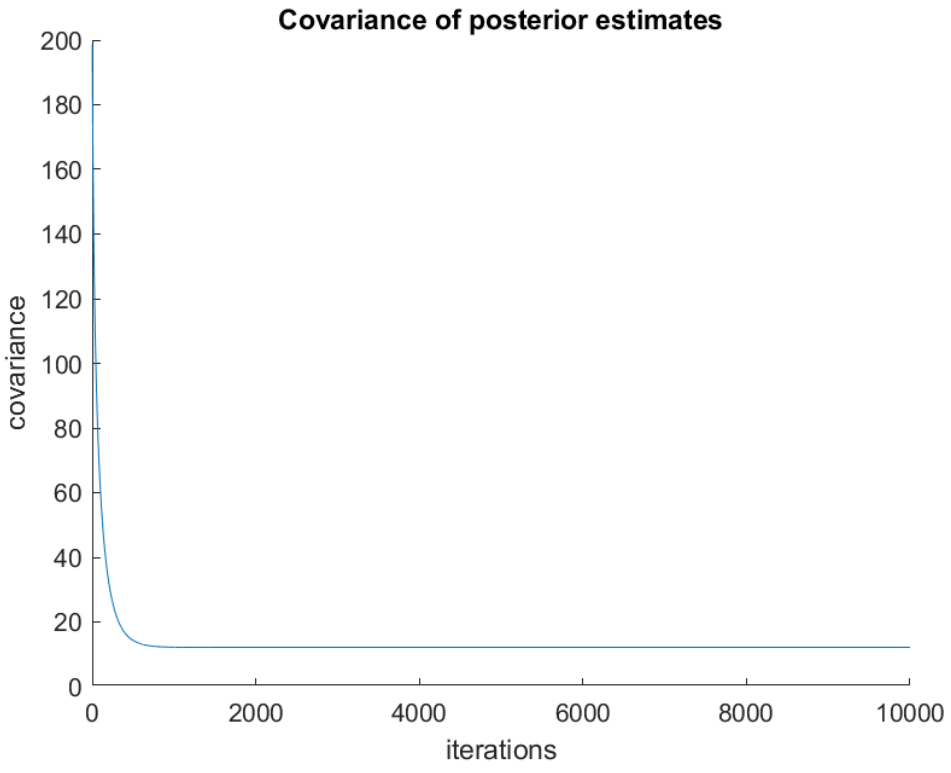
Plots:



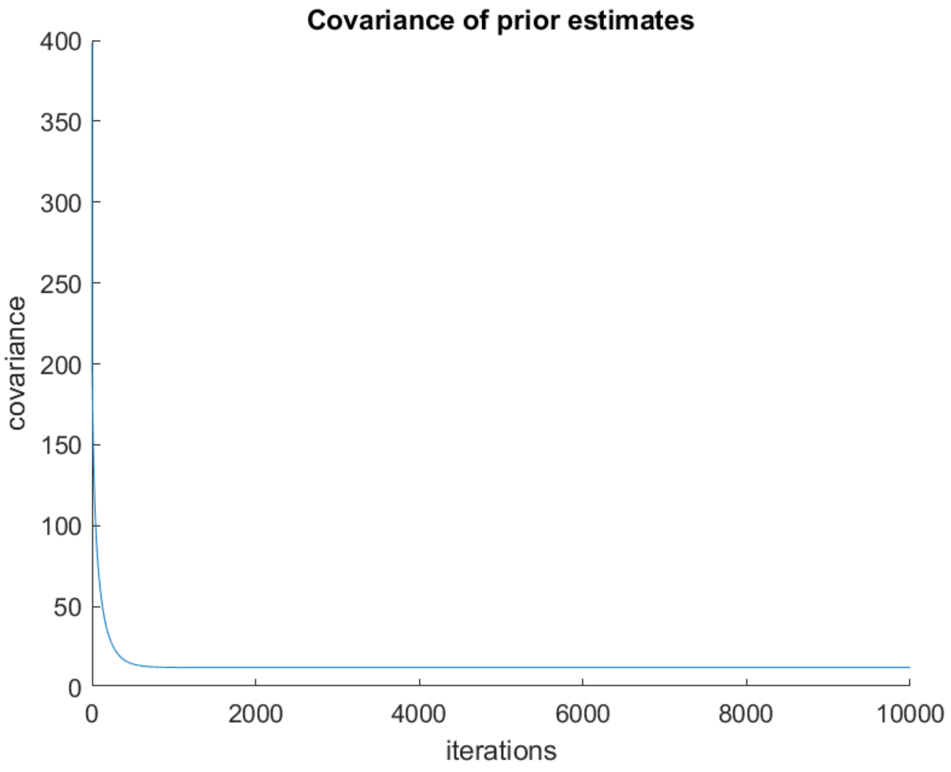
As seen from this plot, the state vector saturates quickly in about 1000 iterations. The final result is not exactly the measured value. This error may be because we have linearised the system, i.e. there is an approximation in the modelling.
The posterior states are those states obtained after the update steps.



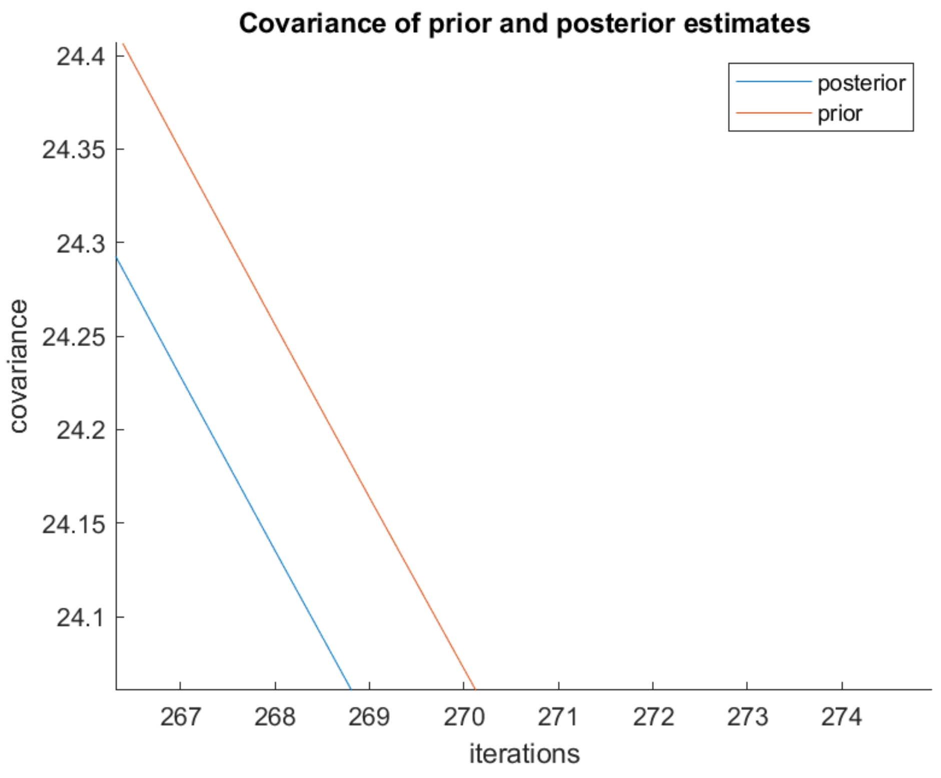
prior state estimates are plotted above. This is the state vector obtained after the prediction step and before the update step.



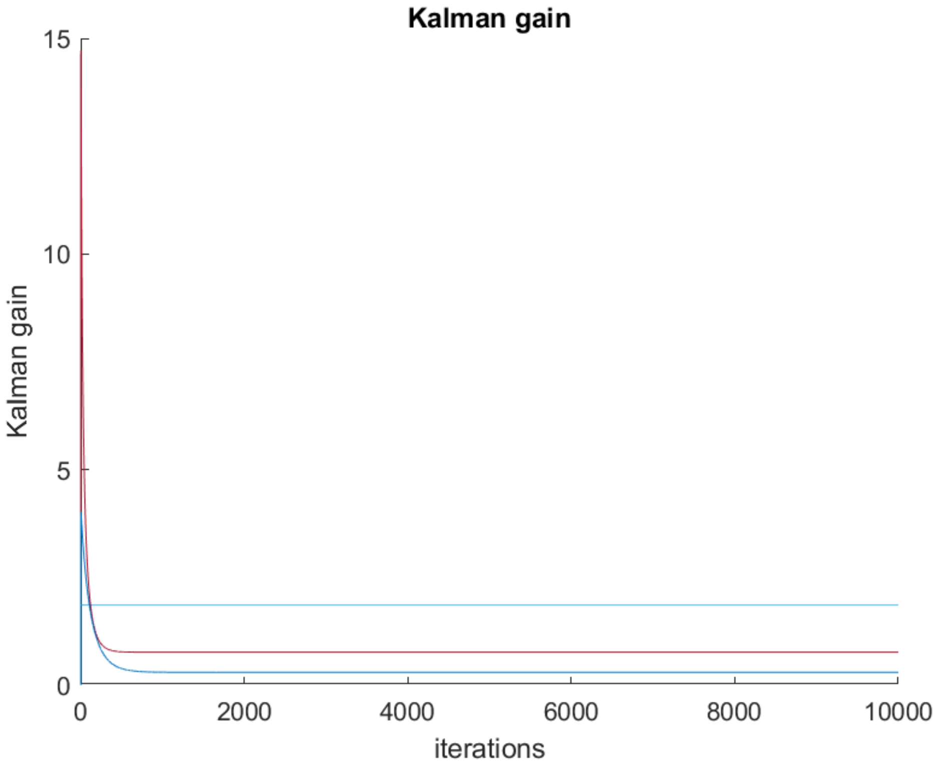
This plot represents the covariance of the posterior estimates. For a converging kalman filter, this value must decay exponentially and saturate at a small value.



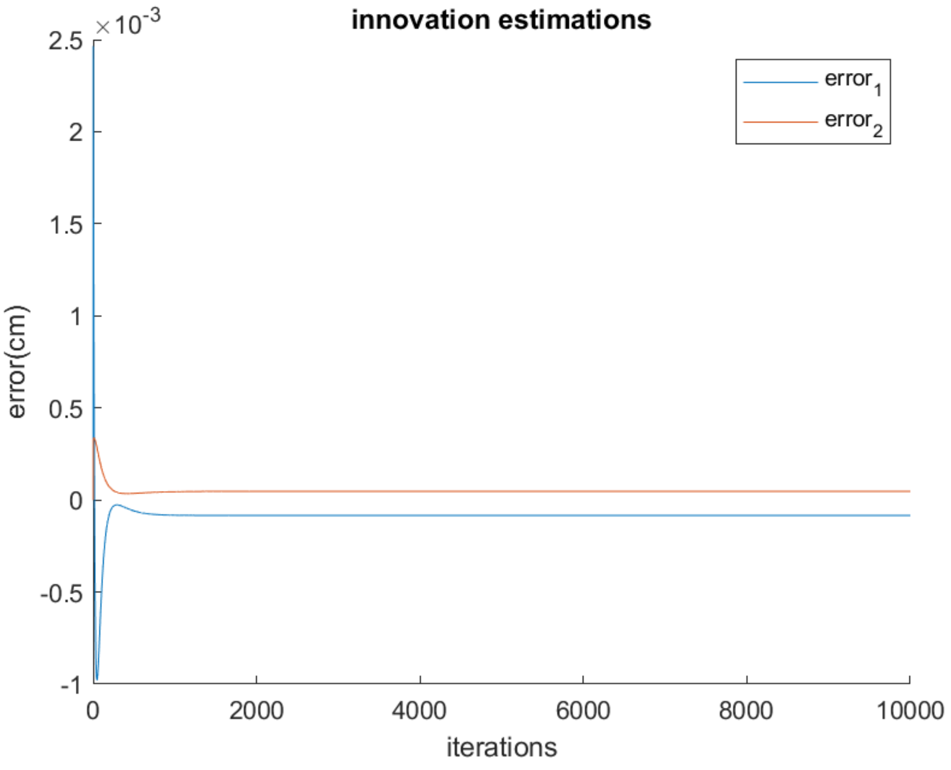
This plot represents the covariance of the prior estimates.



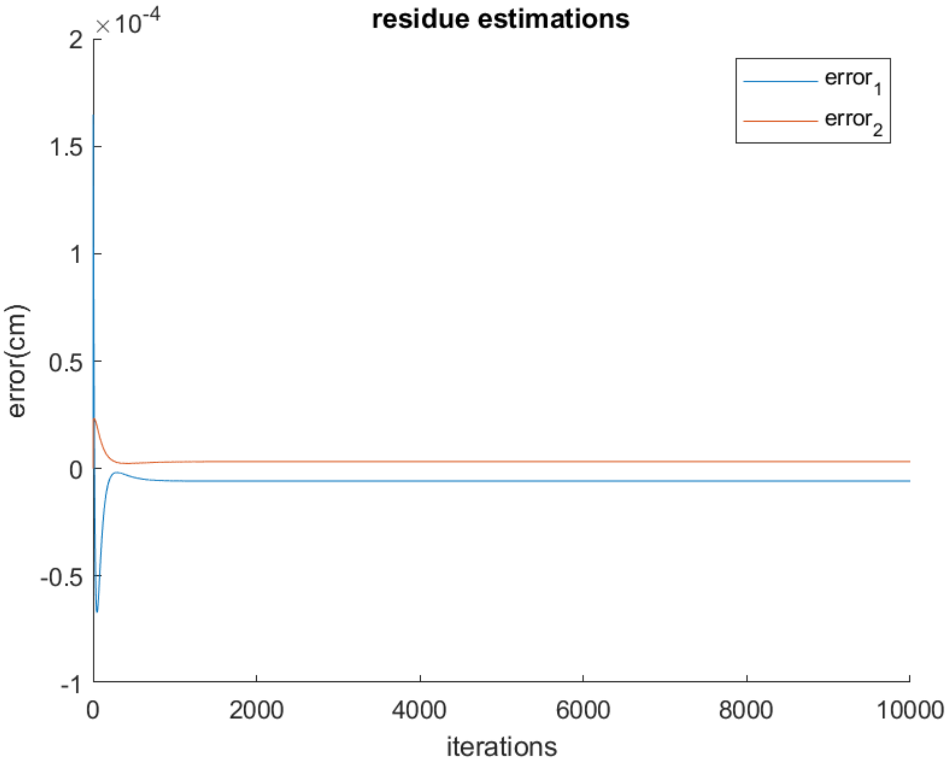
I have plotted both the covariances on the same plot, and I have zoomed into one particular region in the plot to highlight my point.
The covariance of the prior estimate should be greater than the posterior estimate of the previous iteration and the posterior estimate of the same iteration. This is because, when we predict the state at the k- instant, we add our uncertainty in modelling (which would dilate and rotate the ellipse). When we update it, we subtract the Kalman gain from the covariance, reducing the covariance (constricting the ellipse).



Kalman gain is a 4x2 matrix in this question. I have plotted each element separately in this plot. It is observed that some elements of the Kalman gain decay exponentially while others remain constant throughout.



Innovations are errors between the estimated output and the measured values after the prediction step. These values are used to calculate the Kalman gain



Residues are the errors between the estimated output and the measured values after the update step. A functional Kalman filter should have magnitudes of residues to be lesser than the magnitude of the innovations.

Part 2: Particle Filter
Code:
Link to the Particle filter code

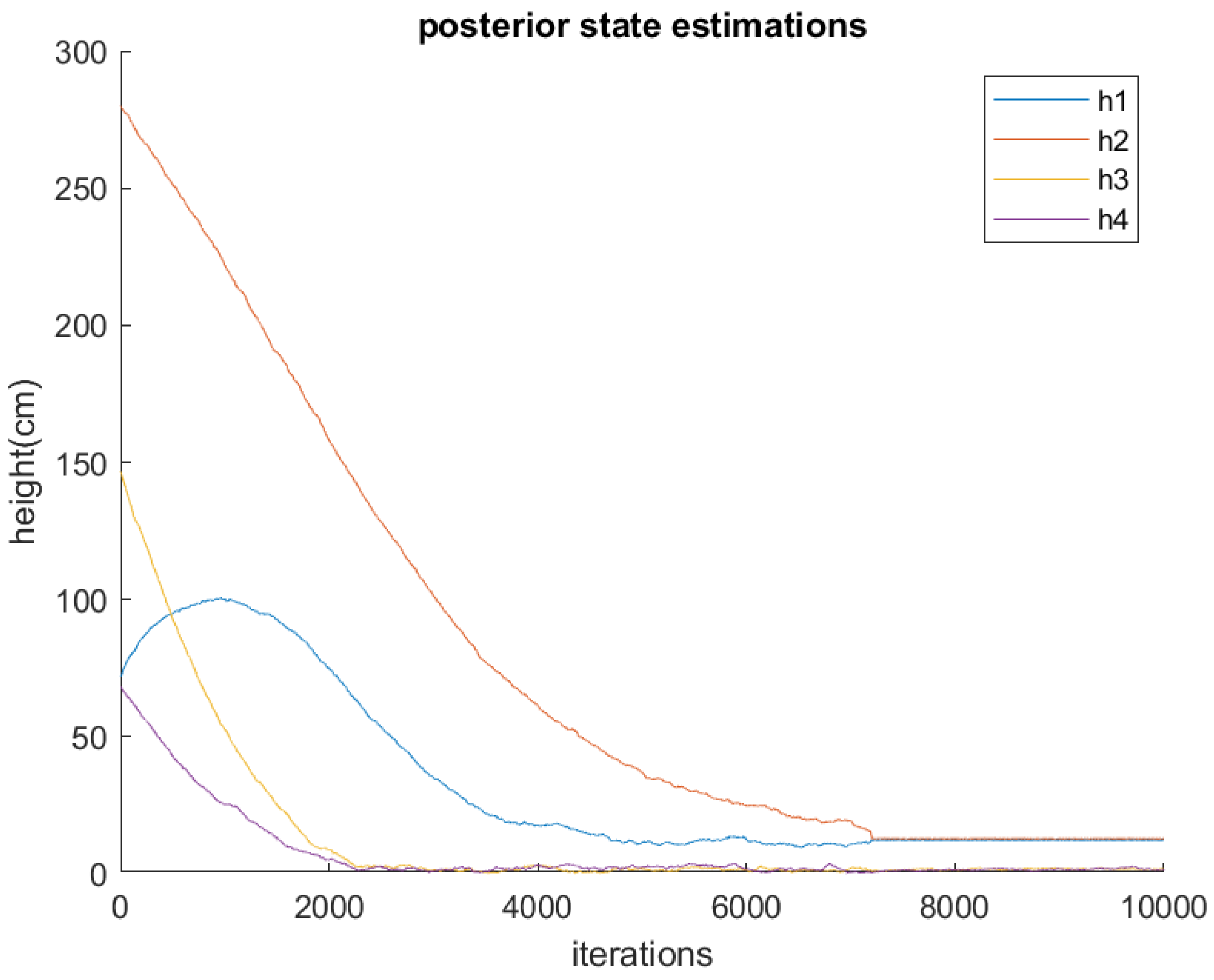
Result:
```
tolerance =

   -0.0049


h =

   12.2599
   12.7804
    1.5669
    1.3860
```

Code:
```
x0 = [12.4;12.7;1.8;1.4];
x0 = x0*ones(1,N) + L*randn(4,N);
x1_post = x0(1,:);
x2_post = x0(2,:);
x3_post = x0(3,:);
x4_post = x0(4,:);
Q = 0.005*eye(4);
w = chol(Q)*randn(4,N);
w1 = w(1,:);
w2 = w(2,:);
w3 = w(3,:);
w4 = w(4,:);
for iter = 1:10000
   %modelling noise
   w = chol(Q)*randn(4,N);
   w1 = w(1,:);
   w2 = w(2,:);
   w3 = w(3,:);
   w4 = w(4,:);
   %predicting step
   for i = 1:N
        x1_pri(i) = x1_post(i) + 0.1*(-a1/A1*sqrt(2*g*x1_post(i)) +
a3/A1*sqrt(2*g*x3_post(i)) + (gamma1*k1*v1)/A1) + w1(i);
       x2_pri(i) = x2_post(i) + 0.1*(-a2/A2*sqrt(2*g*x2_post(i)) +
a4/A2*sqrt(2*g*x4_post(i)) + (gamma2*k1*v2)/A2) + w2(i);
       x3_pri(i) = x3_post(i) + 0.1*(-a3/A3*sqrt(2*g*x3_post(i)) +
(1-gamma2)*k2*v2/A3) + w3(i);
       x4_pri(i) = x4_post(i) + 0.1*(-a4/A4*sqrt(2*g*x4_post(i)) +
(1-gamma1)*k1*v1/A4) + w4(i);
```

```
   end
   x1_pri = abs(x1_pri);
   x2_pri = abs(x2_pri);
   x3_pri = abs(x3_pri);
   x4_pri = abs(x4_pri);

   %finding the estimated output
   z1 = 0.5.*numbers(iter,1);
   z2 = 0.5.*numbers(iter,2);
   z = [z1;z2];
   z_true = z*ones(1,N);
   z_est = [0.5*x1_pri; 0.5*x2_pri];
   v = z_true-z_est;
%finding the weights
   for i = 1:N
       q(i) = double(exp(-0.5*(v(:,i)'*inv(R)*v(:,i))));
   end
   s = sum(q);
   for i = 1:N
       q(i)/s;
       wt(i) = q(i)/s;

   end
   M = length(wt);
   Q1 = cumsum(wt);
   indx = zeros(1,N);
%importance sampling
   T = linspace(0,1-1/N,N);
   i=1; j=1;
   while(i<=N && j<M)
       while Q1(j)<T(i)
           j = j+1;
       end
       indx(i) = j;

       x1_post(i) = x1_pri(j);
       x2_post(i) = x2_pri(j);
       x3_post(i) = x3_pri(j);
       x4_post(i) = x4_pri(j);
       i = i+1;
   end

   x1_pri_cum = cumsum(x1_pri);
   x2_pri_cum = cumsum(x2_pri);
   x3_pri_cum = cumsum(x3_pri);
   x4_pri_cum = cumsum(x4_pri);

   x1_post_cum = cumsum(x1_post);
   x2_post_cum = cumsum(x2_post);
   x3_post_cum = cumsum(x3_post);
   x4_post_cum = cumsum(x4_post);
```

```
    x1_est(iter) = mean(x1_post);
    x2_est(iter) = mean(x2_post);
    x3_est(iter) = mean(x3_post);
    x4_est(iter) = mean(x4_post);
```

Plot:



The particle filter converges, and we get the required state vector values. I have plotted the state estimates as a function of iterations. It can be seen to converge after around 7000 iterations.

Comparison between the Kalman filter and the Particle filter:
The particle filter takes more iterations to converge than the Kalman filter, as seen from the plots above.

	h1	h2	h2	h3
Measured Values	12.26297	12.78316	1.633941	1.409045
Kalman Filter	12.2630	12.7832	1.7851	1.4043
Particle Filter	12.2599	12.7804	1.5669	1.3860

The Kalman filter was implemented after linearising the differential equations defining the model.
On the other hand, the particle filter directly uses the differential equation to predict the prior state estimate.
Implementing a particle filter needs much more computational power, as we have to treat each particle as an ellipse with a centroid and covariance and reduce it as much as possible.
On the other hand, the Kalman filter needs very little computation power.
For Gaussian noise, it can be seen that the Kalman filter gives better results than the particle filter. When the noise is non-Gaussian, then the particle filter will give better results.