

Homework 6 Report

AERO626, Spring 2023

Name: David van Wijk UIN: 932001896

Problem 1 Results

The derivation for problem 1 can be found at the end of this report, after the MATLAB code.

Problem 2 Results

Part A

In Figure 1, the true trajectory and measurements for one set of randomly generated values versus the measurement number is plotted. Here, the random truth is generated at $k = 0$ and measurements are simulated for $k > 0$, up to $k = 500$. The code to generate this plot, along with the plots that follow in this report, can be found in Appendix A.



Figure 1: True trajectory and measurements versus measurement number.

Part B

In order to implement the extended Kalman Filter (EKF), the state and measurement Jacobians must first be computed. Using a first order Taylor series expansion about the mean, these Jacobians can be computed as follows:

$$\mathbf{F}_x(t) = \left. \frac{\partial \mathbf{f}(\mathbf{x}(t), \mathbf{w}(t))}{\partial \mathbf{x}(t)} \right|_{\substack{\mathbf{x}(t) = \mathbf{m}_x(t) \\ \mathbf{w}(t) = \mathbf{0}_w}} \quad (1a)$$

$$\mathbf{F}_w(t) = \left. \frac{\partial \mathbf{f}(\mathbf{x}(t), \mathbf{w}(t))}{\partial \mathbf{w}(t)} \right|_{\substack{\mathbf{x}(t)=\mathbf{m}_x(t) \\ \mathbf{w}(t)=\mathbf{0}_w}} \quad (1b)$$

$$\mathbf{H}_x = \left. \frac{\partial \mathbf{h}(\mathbf{x}(t), \mathbf{v}(t))}{\partial \mathbf{x}(t)} \right|_{\substack{\mathbf{x}(t)=\mathbf{m}_x(t) \\ \mathbf{v}(t)=\mathbf{0}_v}} \quad (2a)$$

$$\mathbf{H}_v = \left. \frac{\partial \mathbf{h}(\mathbf{x}(t), \mathbf{v}(t))}{\partial \mathbf{v}(t)} \right|_{\substack{\mathbf{x}(t)=\mathbf{m}_x(t) \\ \mathbf{v}(t)=\mathbf{0}_v}} \quad (2b)$$

where $\mathbf{f}(\mathbf{x}(t))$ describes the dynamics of the system, and $\mathbf{h}(\mathbf{x}(t))$ is the measurement function. For this specific problem,

$$\mathbf{F}_x = 1 - .01 \cos(x_{k-1}) \quad (3a)$$

$$\mathbf{F}_w = 1 \quad (3b)$$

$$\mathbf{H}_x = \cos(2x_k) \quad (4a)$$

$$\mathbf{H}_v = 1 \quad (4b)$$

In Figure 2, the state estimation error of the EKF versus measurement number is plotted. The 3σ interval of state covariance is also plotted. Note that both the prior and posterior errors and covariances are plotted. The state error remains within the 3σ interval for the entirety of the simulation run. While it is acceptable for the state error to occasionally walk outside of the 3σ interval, doing so consistently is a sign that the estimator may not be performing well. However, this is not a concern here.

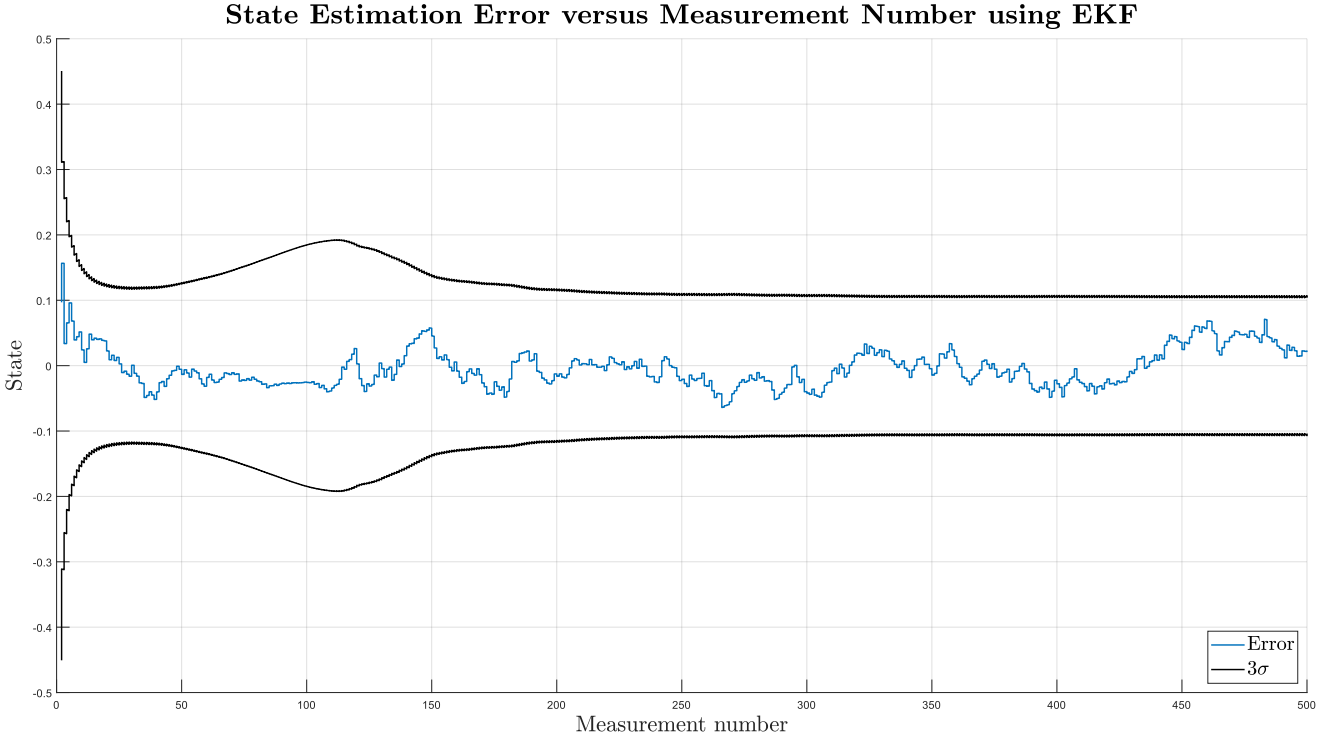


Figure 2: Prior and Posterior Estimation Error for each state versus Measurement Number.

In Figure 3, the innovations are plotted, along with the 3σ intervals of innovation covariance and measurement noise covariance. As expected, the innovation covariance and the measurement noise covariance approach very similar values as all the data is processed.

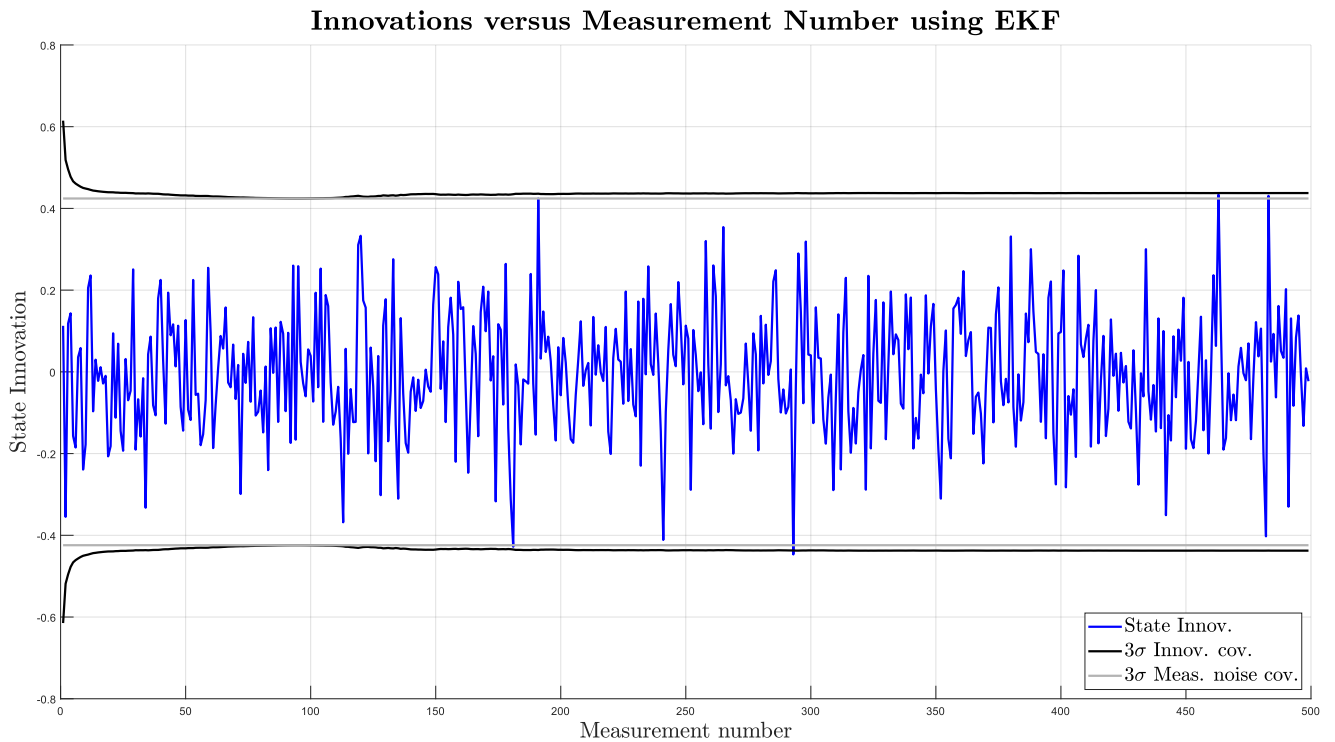


Figure 3: State innovations versus Measurement Number.

Part C

UKF Variation 1

This section considers the results for the Unscented Kalman Filter (UKF) with parameters $\alpha=1$, $\beta = 0$, and $\kappa = 2$. Cholesky decomposition was used for the unscented transform.

In Figure 4, the state estimation error of the first variation of the UKF versus measurement number can be found. The 3σ interval of state covariance is also plotted. Note that both the prior and posterior errors and covariances are plotted. The state error remains within the 3σ interval for the entirety of the simulation run. Again, while it is acceptable for the state error to occasionally walk outside of the 3σ interval, doing so consistently is a sign that the estimator may not be performing well. However, this is not a concern here.

State Estimation Error versus Measurement Number using UFK with parameters: $\alpha=1$ $\kappa=2$ $\beta=0$

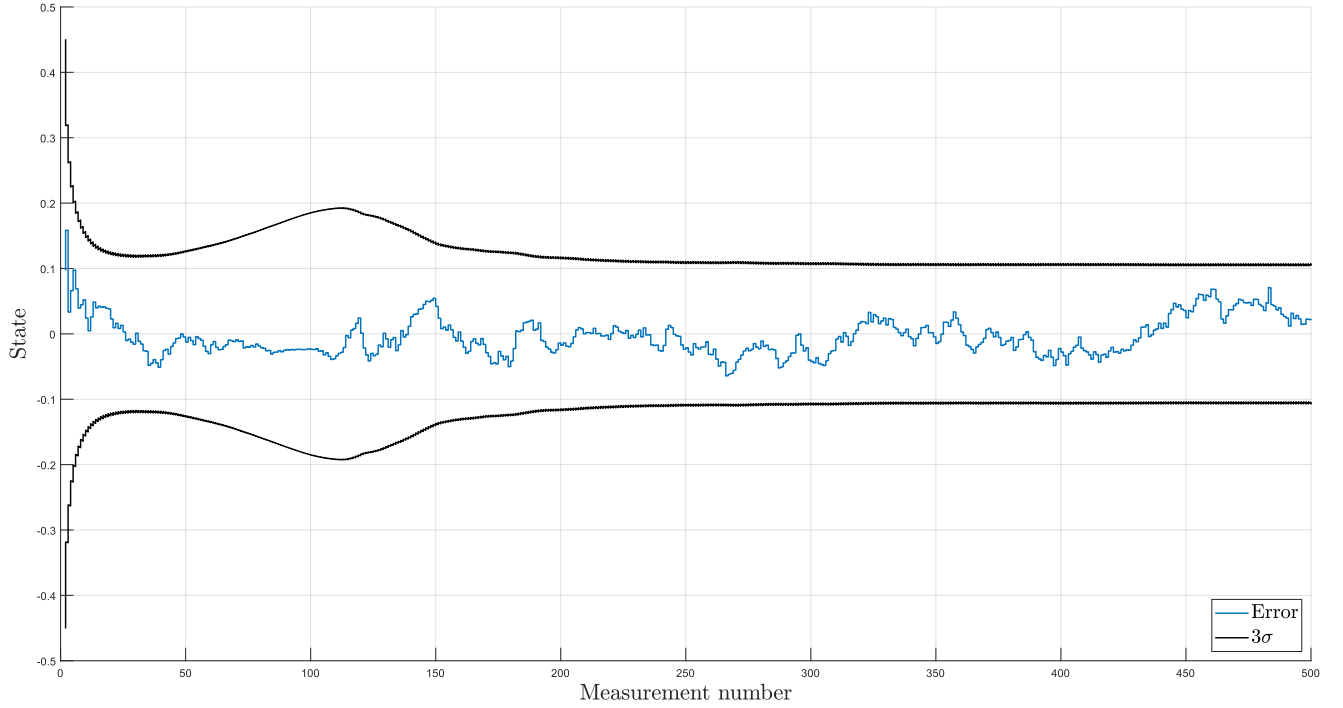


Figure 4: Prior and Posterior Estimation Error for each state versus Measurement Number.

In Figure 5, the innovations are plotted, along with the 3σ intervals of innovation covariance and measurement noise covariance. As expected, the innovation covariance and the measurement noise covariance approach very similar values as all the data is processed.

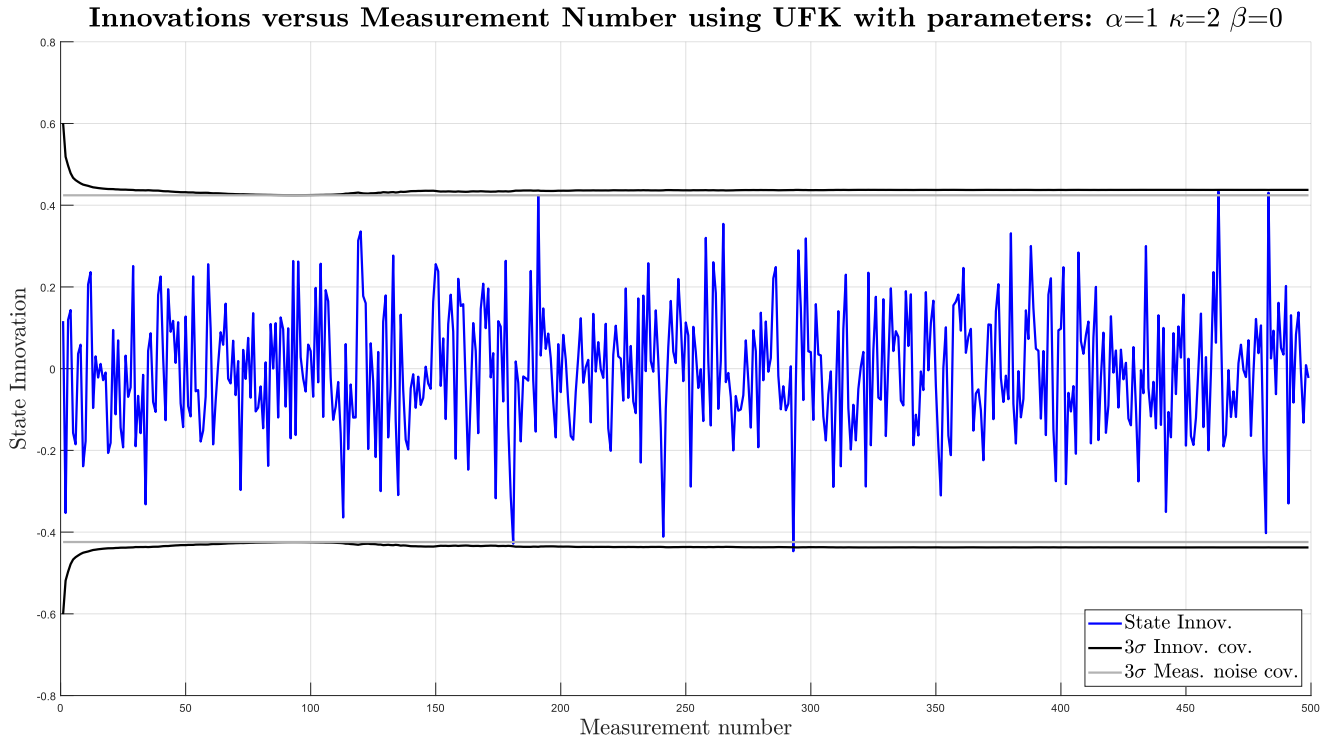


Figure 5: State innovations versus Measurement Number.

UKF Variation 2

This section considers the results for the UKF with parameters $\alpha=0.5$, $\beta = 2$, and $\kappa = 2$. Again, Cholesky decomposition was used for the unscented transform.

In Figure 6, the state estimation error of the second variation of the UKF versus measurement number can be found. The 3σ interval of state covariance is also plotted. Note that both the prior and posterior errors and covariances are plotted. The state error remains within the 3σ interval for the entirety of the simulation run. The state error stays well within the 3σ interval throughout the process, which is a fair visual indicator that the filter is functioning well.

State Estimation Error versus Measurement Number using UKF with parameters: $\alpha=0.5$ $\kappa=2$ $\beta=2$

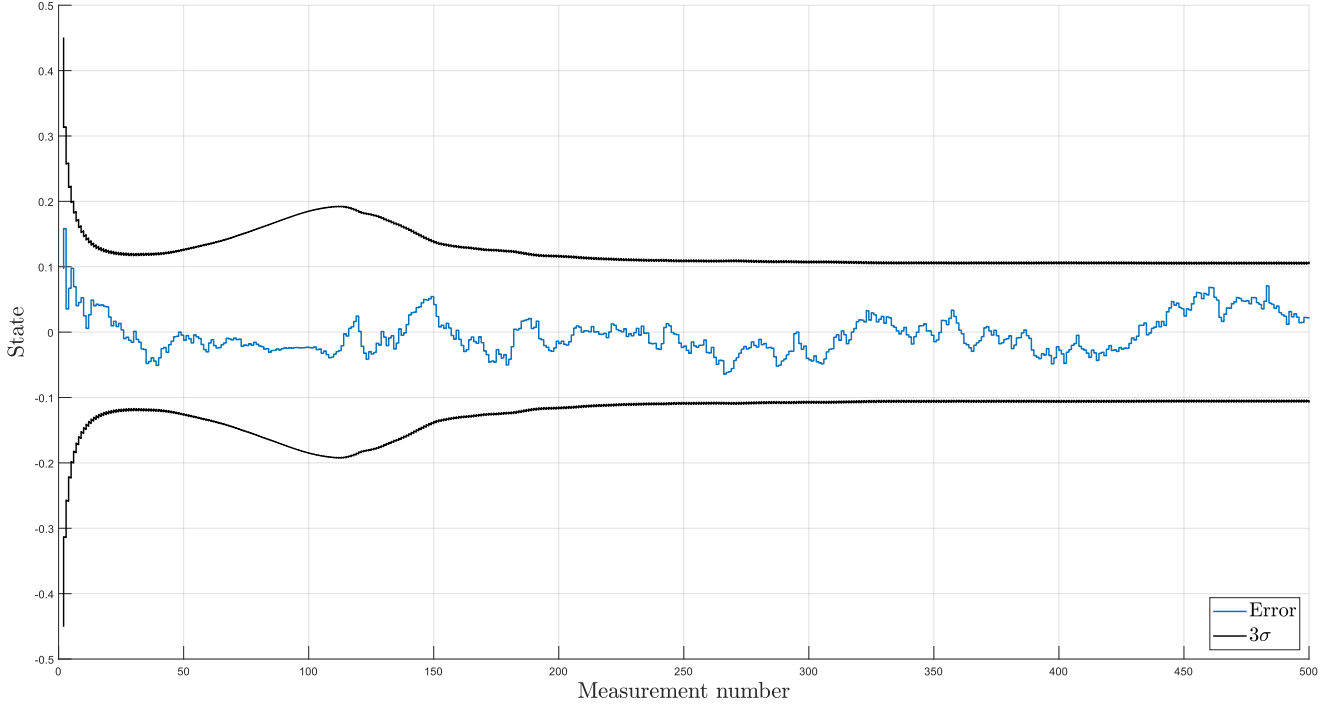


Figure 6: Prior and Posterior Estimation Error for each state versus Measurement Number.

In Figure 7, the innovations are plotted, along with the 3σ intervals of innovation covariance and measurement noise covariance. As expected, the innovation covariance and the measurement noise covariance approach very similar values as all the data is processed.

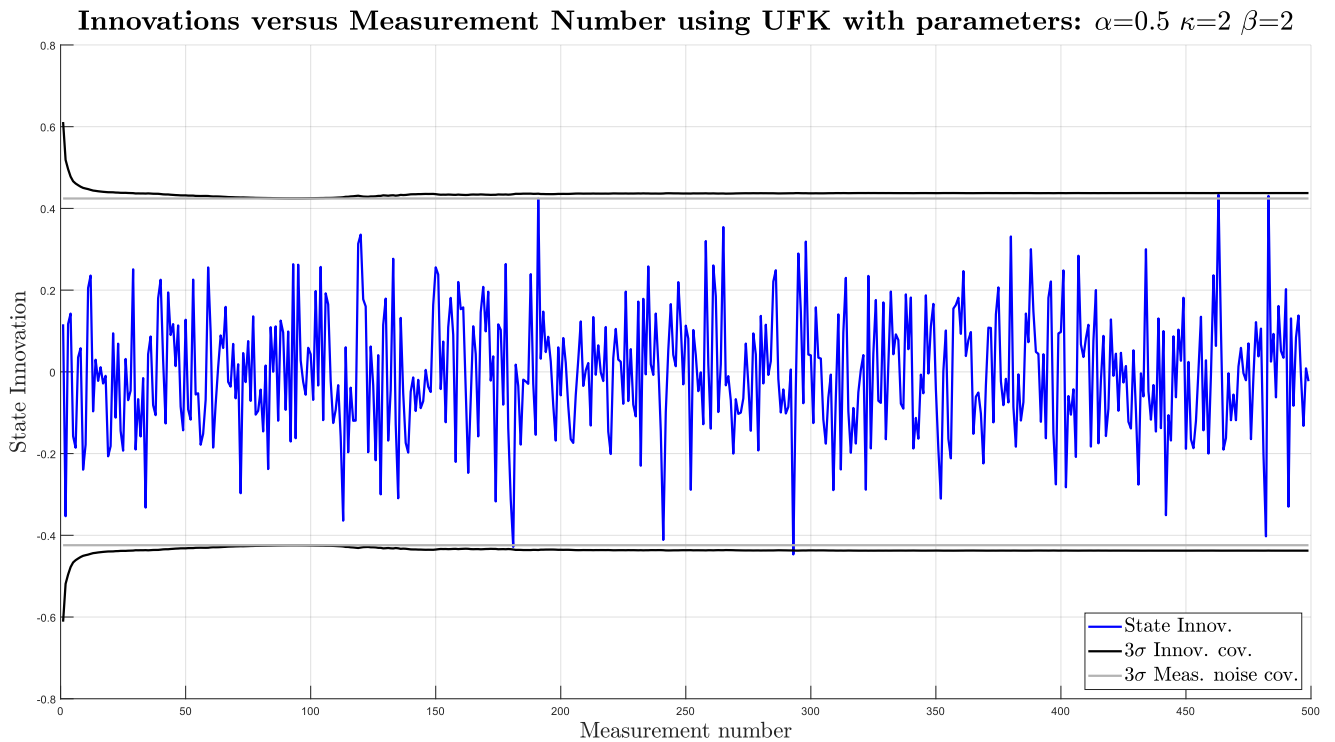


Figure 7: State innovations versus Measurement Number.

Part D

Visually, the three filter variations are nearly identical. However, their performances can be analyzed more rigorously using root mean square (RMS) of the measurement error, as well as the Mahalanobis distance. The RMS of the measurement error is evaluated at each measurement step and is averaged over the number of measurements. This is done for each filter. The RMS is tabularized in Table 1 for each filter variation. It can be observed that the RMS converges to a value that is very close to the measurement uncertainty, obtained by taking the square root of the diagonal elements of the measurement noise covariance. In this case P_{vv} is constant throughout the process. This is a good indicator that all three filters are performing well.

Table 1: Root Mean Square of Error for each filter variation.

Filter Variation	RMS
EKF	0.1488156
UKF 1	0.1487885
UKF 2	0.1487877

Additionally, the Mahalanobis distance at each measurement step is plotted in Figure 8, for each filter variation. This is a scalar measure of how many multi-dimensional standard deviations away from the true state that our estimate is. For this case, since the state vector has a length of 1, a Mahalanobis distance of 1 is a good rule of thumb for adequate performance of the estimator. Therefore we can see that our estimators do a fairly decent job with the data they are given as the Mahalanobis distance stays relatively low throughout the processing of measurements for all three filters. For the majority of the process the Mahalanobis distance is practically the same for all three filters, but there are some instances where the UKF variations have a slightly lower Mahalanobis distance than the EKF, at around 60 to 100 measurements in.

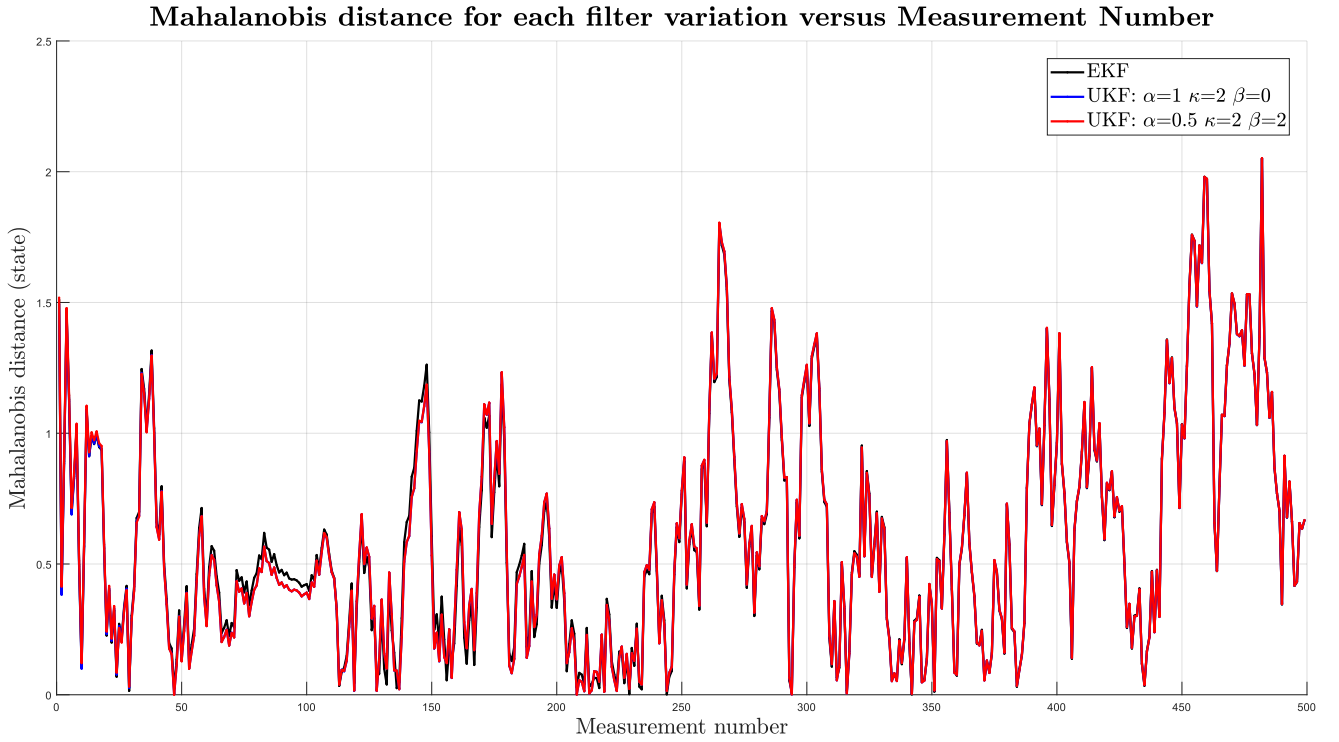


Figure 8: Mahalanobis distance in the state versus Measurement Number for each filter variation.

Overall, all three filters perform quite well, and have similar performances.

Part E

A Monte Carlo simulation was done for $N = 500$ trials using the EKF. The average filter error and the 3σ average filter state estimation error covariance is plotted on Figure 9. These values were calculated by averaging the state estimation error and state uncertainty at each measurement step over the Monte Carlo runs. In addition, the 3σ of Monte Carlo state estimation error covariance is plotted on the same figure. The average state estimation error over a large number of trials for an unbiased estimator should approach zero. From the figure, it appears that this filter is unbiased since the average state estimation remains quite close to zero. Additionally, 3σ of the Monte Carlo state estimation error covariance is consistently less than the 3σ average filter state estimation error covariance for most of the process, which makes sense.

State Estimation Error versus Measurement Number using EKF, Monte Carlo Analysis with: 500 Trials

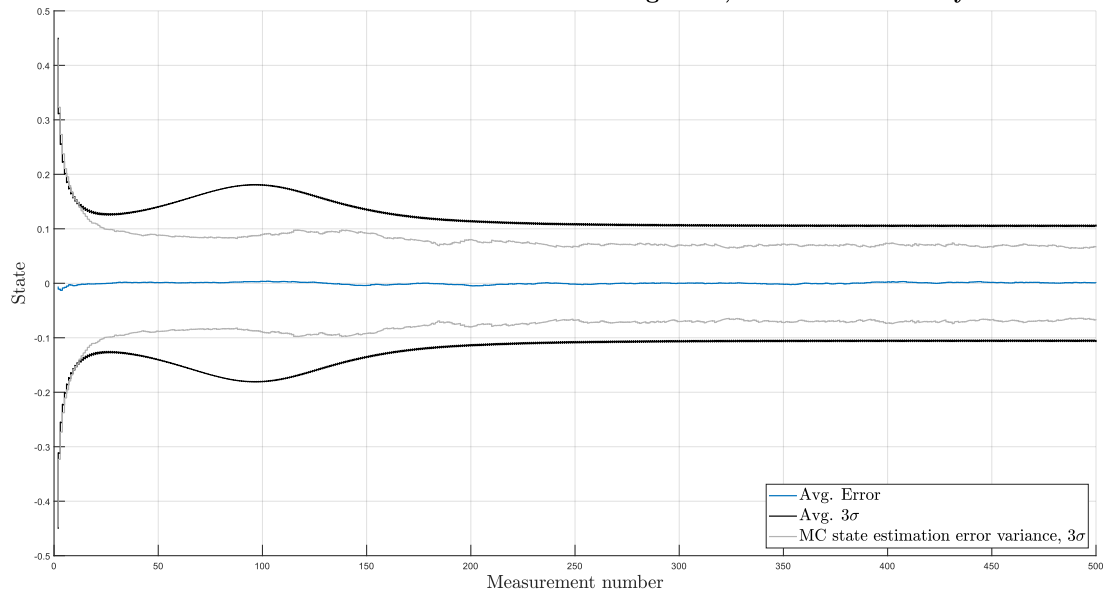


Figure 9: Average filter state error and state covariance Monte Carlo state estimation error variance for Monte Carlo simulation of $N = 500$.

A Matlab Code

```
%% AERO 626 Homework 6 - Problem 2
%
%   Texas A&M University
%   Aerospace Engineering
%   van Wijk, David

close all; clear; clc;
plot_flag = true;

% Seed to reproduce results
rng(10)

xaxis_sz = 20; yaxis_sz = 20; legend_sz = 18;

%% Part A - Initialization and Data Generation

Pww = .01^2;
Pvv = .02;
mx0 = 1.5;
Pxx0 = .15^2;
numPts = 500;

% Generate random truth
x0 = mx0 + randn*sqrt(Pxx0);

x_truth = recursivePropFull(x0,0,numPts);
z_full = measurementFunFull(x_truth,Pvv,numPts);

if plot_flag
    plotPartA(x_truth,z_full,xaxis_sz,yaxis_sz,legend_sz) %#ok<*UNRCH>
end

%% Part B - EKF Implementation

opts = odeset('AbsTol',1e-9,'RelTol',1e-9);

% NOTATION:
%   tkm1    = t_{k-1}           time at the (k-1)th time
%   mxkm1   = m_{x,k-1}^{+}     posterior mean at the (k-1)th time
%   Pxxkm1  = P_{xx,k-1}^{+}    posterior covariance at the (k-1)th time
%   tk      = t_{k}            time at the kth time
%   mxkm    = m_{x,k}^{-}       prior mean at the kth time
%   Pxxkm   = P_{xx,k}^{-}      prior covariance at the kth time
%   mxkp    = m_{x,k}^{+}       posterior mean at the kth time
%   Pxxkp   = P_{xx,k}^{+}      posterior covariance at the kth time
%   Pzzkm   = P_{zz,k}^{-}      innovation covariance at the kth time
%   Pvvk    = P_{vv,k}         measurement noise covariance at the kth time

% declare storage space for saving state estimation error information
xcount = 0;
xstore = nan(1,2*numPts-1);
```

```

kxstore = nan(1,2*numPts-1);
mxstore = nan(1,2*numPts-1);
exstore = nan(1,2*numPts-1);
sxstore = nan(1,2*numPts-1);

% declare storage space
zcount = 0;
zstore = nan(1,numPts);
kzstore = nan(1,numPts);
mzstore = nan(1,numPts);
ezstore = nan(1,numPts);
dzstore = nan(1,numPts);
dxstore = nan(1,numPts);
szstore = nan(1,numPts);
svstore = nan(1,numPts);

% store initial data
xcount = xcount + 1;
kxstore(:,xcount) = 0;
mxstore(:,xcount) = mx0;
sxstore(:,xcount) = sqrt(diag(Pxx0));

% measurement noise
Hv = 1;

% process noise
Fw = 1;

% initialize time, mean, and covariance for the EKF
mxkm1 = mx0;
Pxxkm1 = Pxx0;

% loop over the number of data points
for k = 2:numPts
    zk = z_full(k,:); % current measurement to process
    Pvvk = Pvv; % measurement noise covariance

    % unpack the truth -- this cannot be used in the filter, only for
    % analysis
    xk = x_truth(k,:); % true state

    % propagate the mean and covariance
    mxkm = recursivePropSingle(mxkm1,0);
    Fx = stateJacobianMean(mxkm1);
    Pxxkm = Fx*Pxxkm1*Fx' + Fw*Pww*Fw';

    % store a priori state information for analysis
    xcount = xcount + 1;
    xstore(:,xcount) = xk;
    kxstore(:,xcount) = k;
    mxstore(:,xcount) = mxkm;
    exstore(:,xcount) = xk - mxkm;
    sxstore(:,xcount) = sqrt(diag(Pxxkm));

```

```

% compute the estimated measurement
mzkm = measurementFunSingle(mxkm,0);

% compute the measurement Jacobian
Hxk = measurementJacobianMean(mxkm);

% update the mean and covariance
Pxzkm = Pxxkm*Hxk';
Pzzkm = Hxk*Pxxkm*Hxk' + Hv*Pvvk*Hv';
Kk = Pxzkm/Pzzkm;
mxkp = mxkm + Kk*(zk - mzkm);
Pxxkp = Pxxkm - Pxzkm*Kk' - Kk*(Pxzkm)' + Kk*(Pzzkm)*Kk';

% store a posteriori state information for analysis
xcount = xcount + 1;
xstore(:,xcount) = xk;
kxstore(:,xcount) = k;
mxstore(:,xcount) = mxkp;
exstore(:,xcount) = xk - mxkp;
sxstore(:,xcount) = sqrt(diag(Pxxkp));

% store measurement information for analysis
zcount = zcount + 1;
zstore(:,zcount) = zk;
kzstore(:,zcount) = k;
mzstore(:,zcount) = mzkm;
ezstore(:,zcount) = zk - mzkm;
dzstore(:,zcount) = (zk - mzkm)'*(Pzzkm\'(zk - mzkm));
dxstore(:,zcount) = (xk - mxkp)'*(Pxxkp\'(xk - mxkp));
szstore(:,zcount) = sqrt(diag(Pzzkm));
svstore(:,zcount) = sqrt(diag(Pvvk));

% cycle the time, mean, and covariance for the next step of the EKF
mxkm1 = mxkp;
Pxxkm1 = Pxxkp;
end

dxstoreEKF = dxstore;
rmsEKF = sqrt((sum(ezstore(1:end-1).^2))/length(ezstore(1:end-1)));

if plot_flag
    % Plot Innovations
    plotPartB_Innovations(ezstore,szstore,svstore,xaxis_sz,yaxis_sz,
        legend_sz)

    % Plot Estimation Error
    plotPartB_EstimationError(exstore,sxstore,numPts,xaxis_sz,yaxis_sz,
        legend_sz)
end

%% Part C - UKF Implementation

```

```

% UKF parameters
params = {[1, 2, 0],[0.5, 2, 2]};

for i = 1:length(params)
    params_curr = params(1,i);
    alpha = params_curr{1}(1);
    kappa = params_curr{1}(2);
    beta = params_curr{1}(3);

    % NOTATION:
    %   tkm1    = t_{k-1}           time at the (k-1)th time
    %   mxkm1   = m_{x,k-1}^{+}    posterior mean at the (k-1)th time
    %   Pxxkm1  = P_{xx,k-1}^{+}   posterior covariance at the (k-1)th time
    %   tk      = t_{k}            time at the kth time
    %   mxkm    = m_{x,k}^{-}      prior mean at the kth time
    %   Pxxkm   = P_{xx,k}^{-}     prior covariance at the kth time
    %   mxkp    = m_{x,k}^{+}      posterior mean at the kth time
    %   Pxxkp   = P_{xx,k}^{+}     posterior covariance at the kth time
    %   Pzzkm   = P_{zz,k}^{-}     innovation covariance at the kth time
    %   Pvvk    = P_{vv,k}         measurement noise covariance at the kth
    %   time

    % declare storage space for saving state estimation error information
    xcount = 0;
    xstore = nan(1,2*numPts-1);
    kxstore = nan(1,2*numPts-1);
    mxstore = nan(1,2*numPts-1);
    exstore = nan(1,2*numPts-1);
    sxstore = nan(1,2*numPts-1);

    % declare storage space
    zcount = 0;
    zstore = nan(1,numPts);
    kzstore = nan(1,numPts);
    mzstore = nan(1,numPts);
    ezstore = nan(1,numPts);
    dzstore = nan(1,numPts);
    dxstore = nan(1,numPts);
    szstore = nan(1,numPts);
    svstore = nan(1,numPts);

    % store initial data
    xcount = xcount + 1;
    kxstore(:,xcount) = 0;
    mxstore(:,xcount) = mx0;
    sxstore(:,xcount) = sqrt(diag(Pxx0));

    % measurement noise
    Hv = 1;

    % process noise
    Fw = 1;

```

```

% initialize time, mean, and covariance for the EKF
mxkm1 = mx0;
Pxxkm1 = Pxx0;

% loop over the number of data points
for k = 2:numPts
    zk = z_full(k,:); % current measurement to process
    Pvvk = Pvv; % measurement noise covariance

    % unpack the truth -- this cannot be used in the filter, only for
    % analysis
    xk = x_truth(k,:); % true state

    % propagate the mean and covariance
    [mxkm, Pxxkm, ~] = UT(mxkm1, Pxxkm1, @recursivePropSingle, alpha,
        kappa, beta, 0);
    Pxxkm = Pxxkm + Pww;

    % store a priori state information for analysis
    xcount = xcount + 1;
    xstore(:, xcount) = xk;
    kxstore(:, xcount) = k;
    mxstore(:, xcount) = mxkm;
    exstore(:, xcount) = xk - mxkm;
    sxstore(:, xcount) = sqrt(diag(Pxxkm));

    % update the mean and covariance
    [mzkm, Pzzkm, Pxzkm] = UT(mxkm, Pxxkm, @measurementFunSingle, alpha,
        kappa, beta, 0);
    Pzzkm = Pzzkm + Pvvk;

    Kk = Pxzkm/Pzzkm;
    mxkp = mxkm + Kk*(zk - mzkm);
    Pxxkp = Pxxkm - Pxzkm*Kk' - Kk*(Pxzkm)' + Kk*(Pzzkm)*Kk';

    % store a posteriori state information for analysis
    xcount = xcount + 1;
    xstore(:, xcount) = xk;
    kxstore(:, xcount) = k;
    mxstore(:, xcount) = mxkp;
    exstore(:, xcount) = xk - mxkp;
    sxstore(:, xcount) = sqrt(diag(Pxxkp));

    % store measurement information for analysis
    zcount = zcount + 1;
    zstore(:, zcount) = zk;
    kzstore(:, zcount) = k;
    mzstore(:, zcount) = mzkm;
    ezstore(:, zcount) = zk - mzkm;
    dzstore(:, zcount) = (zk - mzkm)'*(Pzzkm\((zk - mzkm)));
    dxstore(:, zcount) = (xk - mxkp)'*(Pxxkp\((xk - mxkp)));
    szstore(:, zcount) = sqrt(diag(Pzzkm));
    svstore(:, zcount) = sqrt(diag(Pvvk));

```

```

        % cycle the time, mean, and covariance for the next step of the
        EKF
        mxkm1 = mxkp;
        Pxxkm1 = Pxxkp;
    end

    if i == 1
        dxstoreUKF1 = dxstore;
        rmsUKF1 = sqrt((sum(ezstore(1:end-1).^2))/length(ezstore(1:
            end-1))));
        params1 = params_curr{:};
    elseif i == 2
        dxstoreUKF2 = dxstore;
        rmsUKF2 = sqrt((sum(ezstore(1:end-1).^2))/length(ezstore(1:
            end-1))));
        params2 = params_curr{:};
    end

    if plot_flag
        params_plot = [alpha, kappa, beta];
        % Plot innovations
        plotPartC_Innovations(ezstore,szstore,svstore,xaxis_sz,yaxis_sz,
            legend_sz,params_plot)

        % Plot state estimation error
        plotPartC_EstimationError(exstore,sxstore,numPts,xaxis_sz,
            yaxis_sz,legend_sz,params_plot)
    end
end

%% Part D - Comparison of Filters

MD_EKF = dxstoreEKF;
MD_UKF1 = dxstoreUKF1;
MD_UKF2 = dxstoreUKF2;
plotPartD_MD_state(MD_EKF,MD_UKF1,MD_UKF2,numPts,xaxis_sz,yaxis_sz,
    legend_sz,params1,params2)

format long;
rmsEKF
rmsUKF1
rmsUKF2

%% Part E - Monte Carlo Simulation

N = 500;

filter_state_error_full = ones(N,2*numPts-1);
filter_state_std_full = ones(N,2*numPts-1);

for j = 1:N

```

```

% rng(j)
% Generate random truth
x0 = mx0 + randn*sqrt(Pxx0);

x_truth = recursivePropFull(x0,0,numPts);
z_full = measurementFunFull(x_truth,Pvv,numPts);

% declare storage space for saving state estimation error information
xcount = 0;
xstore = nan(1,2*numPts-1);
kxstore = nan(1,2*numPts-1);
mxstore = nan(1,2*numPts-1);
exstore = nan(1,2*numPts-1);
sxstore = nan(1,2*numPts-1);

% declare storage space
zcount = 0;
zstore = nan(1,numPts);
kzstore = nan(1,numPts);
mzstore = nan(1,numPts);
ezstore = nan(1,numPts);
dzstore = nan(1,numPts);
szstore = nan(1,numPts);
svstore = nan(1,numPts);

% store initial data
xcount = xcount + 1;
kxstore(:,xcount) = 0;
mxstore(:,xcount) = mx0;
sxstore(:,xcount) = sqrt(diag(Pxx0));

% measurement noise
Hv = 1;

% process noise
Fw = 1;

% initialize time, mean, and covariance for the EKF
mxkm1 = mx0;
Pxxkm1 = Pxx0;

% loop over the number of data points
for k = 2:numPts
    zk = z_full(k,:); % current measurement to process
    Pvvk = Pvv; % measurement noise covariance

    % unpack the truth -- this cannot be used in the filter, only for
    % analysis
    xk = x_truth(k,:); % true state

    % propagate the mean and covariance
    mxkm = recursivePropSingle(mxkm1,0);
    Fx = stateJacobianMean(mxkm1);

```

```

Pxxkm = Fx*Pxxkm1*Fx' + Fw*Pww*Fw';

% store a priori state information for analysis
xcount = xcount + 1;
xstore(:,xcount) = xk;
kxstore(:,xcount) = k;
mxstore(:,xcount) = mxkm;
exstore(:,xcount) = xk - mxkm;
sxstore(:,xcount) = sqrt(diag(Pxxkm));

% compute the estimated measurement
mzkm = measurementFunSingle(mxkm,0);

% compute the measurement Jacobian
Hxk = measurementJacobianMean(mxkm);

% update the mean and covariance
Pxzkm = Pxxkm*Hxk';
Pzzkm = Hxk*Pxxkm*Hxk' + Hv*Pvvk*Hv';
Kk = Pxzkm/Pzzkm;
mxkp = mxkm + Kk*(zk - mzkm);
Pxxkp = Pxxkm - Pxzkm*Kk' - Kk*(Pxzkm)' + Kk*(Pzzkm)*Kk';

% store a posteriori state information for analysis
xcount = xcount + 1;
xstore(:,xcount) = xk;
kxstore(:,xcount) = k;
mxstore(:,xcount) = mxkp;
exstore(:,xcount) = xk - mxkp;
sxstore(:,xcount) = sqrt(diag(Pxxkp));

% store measurement information for analysis
zcount = zcount + 1;
zstore(:,zcount) = zk;
kzstore(:,zcount) = k;
mzstore(:,zcount) = mzkm;
ezstore(:,zcount) = zk - mzkm;
dzstore(:,zcount) = (zk - mzkm)'*(Pzzkm\'(zk - mzkm));
szstore(:,zcount) = sqrt(diag(Pzzkm));
svstore(:,zcount) = sqrt(diag(Pvvk));

% cycle the time, mean, and covariance for the next step of the
EKF
mxkm1 = mxkp;
Pxxkm1 = Pxxkp;
end
filter_state_error_full(j,:) = exstore;
filter_state_std_full(j,:) = sxstore;
end
avg_filter_state_error = sum(filter_state_error_full,1)/N;
avg_filter_state_std = sum(filter_state_std_full,1)/N;
MC_variance = var(filter_state_error_full,0,1);
% MC_variance = var(filter_state_std_full,0,1);

```



```
plotPartE_EstimationErrorMC(avg_filter_state_error,avg_filter_state_std,
    MC_variance,N,numPts,xaxis_sz,yaxis_sz,legend_sz)
```

```
%% UKF Functions
```

```
function [m_y,Py,Pxy] = UT(m_x,Pxx,gfun,alpha,kappa,beta,P_noise)
n = length(m_x);
lambda = alpha^2*(n + kappa) - n;
Sxx = chol(Pxx)';
x_bar = repmat(m_x,1,2*n+1) + sqrt(n+lambda)*[zeros(n),Sxx,-Sxx];
omega_m = [lambda/(n+lambda); (1/(2*(n+lambda)))*ones(2*n,1)];
omega_c = [lambda/(n+lambda) + (1-alpha^2+beta); (1/(2*(n+lambda)))*ones(2*n,1)];
```

```
m_y = zeros(1,n);
for i = 1:(2*n+1)
    y_i = gfun(x_bar(:,i),P_noise);
    w_i_m = omega_m(i,:);
    m_y = m_y + w_i_m * y_i;
end
Py = zeros(n,n);
Pxy = zeros(n,n);
for i = 1:(2*n+1)
    y_i = gfun(x_bar(:,i),P_noise);
    x_i = x_bar(:,i);
    w_i_c = omega_c(i,:);
    Py = Py + w_i_c*(y_i - m_y)*(y_i - m_y)';
    Pxy = Pxy + w_i_c*(x_i - m_x)*(y_i - m_y)';
end
end
```

```
%% Dynamics Functions
```

```
function [Fx] = stateJacobianMean(xkminus1)
Fx = 1 - .01*cos(xkminus1);
end
```

```
function [Hx] = measurementJacobianMean(xk)
Hx = cos(2*xk);
end
```

```
function [z] = measurementFunFull(x,Pvv,numMeasurements)
% Generate measurments for k > 0, for all measurements
```

```
z = nan(numMeasurements,1);
for i = 2:numMeasurements
    xk = x(i);
    vk = randn*sqrt(Pvv);
    zk = .5*sin(2*xk) + vk;
    z(i,1) = zk;
end
end
```

```

function [zk] = measurementFunSingle(xk,Pvv)
% Generate measurments for k > 0, for all measurements

vk = randn*sqrt(Pvv);
zk = .5*sin(2*xk) + vk;

end

function [x] = recursivePropFull(x0,Pww,numPts)
% Recursively propagate the state

x = zeros(numPts,1);
x(1) = x0;
for i = 2:numPts
    xkminus1 = x(i-1);
    wkminus1 = randn*sqrt(Pww);
    xk = xkminus1 - .01*sin(xkminus1) + wkminus1;
    x(i) = xk;
end
end

function [xk] = recursivePropSingle(x0,Pww)
% Recursively propagate the state for all measurements

xkminus1 = x0;
wkminus1 = randn*sqrt(Pww);
xk = xkminus1 - .01*sin(xkminus1) + wkminus1;

end

%% Plotting Functions

function plotPartA(x,z,xaxis_sz,yaxis_sz,legend_sz)
measx = 1:length(x);
figure; grid on; set(gcf, 'WindowState', 'maximized'); hold on;
title('\textbf{True Trajectory vs. Measurements}','FontSize',25,'
    interpreter','latex')
b1 = plot(measx,x,"Color",'b','LineWidth',2);
a1 = scatter(measx(2:end),z(2:end),'filled','MarkerFaceColor','r');
ylabel('State','FontSize',yaxis_sz,'interpreter','latex')
xlabel('Measurement number','FontSize',xaxis_sz,'interpreter','latex')
legendtxt = {'True Trajectory', 'Measurements'};
legend([b1 a1],legendtxt,'FontSize',legend_sz,'interpreter','latex','
    location','northwest')
end

function plotPartB_Innovations(ezstore,szstore,svstore,xaxis_sz,yaxis_sz,
    legend_sz)
measx = 1:length(ezstore);
std_plot = 3; txt = [num2str(std_plot) '$\sigma$'];

figure; grid on; set(gcf, 'WindowState', 'maximized'); hold on;

```

```

title('\textbf{Innovations versus Measurement Number using EKF}','
    Fontsize',25,'interpreter','latex')
a1 = plot(measx,ezstore(1,:), '-','Color','b','LineWidth',2,'MarkerSize'
    ,20);
b1 = plot(measx,std_plot*szstore(1,:), '-','Color','k','LineWidth',2,'
    MarkerSize',20);
b2 = plot(measx,std_plot*svstore(1,:), '-','Color',[.7 .7 .7],'LineWidth'
    ,2,'MarkerSize',20);
plot(measx,-std_plot*szstore(1,:), '-','Color','k','LineWidth',2,'
    MarkerSize',20);
plot(measx,-std_plot*svstore(1,:), '-','Color',[.7 .7 .7],'LineWidth',2,'
    MarkerSize',20);
ylabel('State Innovation','FontSize',yaxis_sz,'interpreter','latex')
xlabel('Measurement number','FontSize',xaxis_sz,'interpreter','latex')
legendtxt = {'State Innov.', [txt ' Innov. cov.'],[txt ' Meas. noise cov
    .']};
legend([a1 b1 b2],legendtxt,'FontSize',legend_sz,'interpreter','latex','
    location','southeast')
end

function plotPartB_EstimationError(exstore,sxstore,numPts,xaxis_sz,
    yaxis_sz,legend_sz)
measx = 1:numPts;
std_plot = 3; txt = [num2str(std_plot) '$\sigma$'];

err_line_opts = {'-','LineWidth',1.3};
std_line_opts = {'-','LineWidth',1.3,'Color','k'};

measx1 = sort([measx measx]);

figure; grid on; set(gcf, 'WindowState', 'maximized'); hold on;
title('\textbf{State Estimation Error versus Measurement Number using EKF
    }','FontSize',25,'interpreter','latex')
a1 = plot(measx1(3:end),exstore(1,2:end),err_line_opts{:});
a2 = plot(measx1(3:end),std_plot*sxstore(1,2:end),std_line_opts{:});
plot(measx1(3:end),-std_plot*sxstore(1,2:end),std_line_opts{:})
ylabel('State','FontSize',yaxis_sz,'interpreter','latex')
xlabel('Measurement number','FontSize',xaxis_sz,'interpreter','latex')
legendtxt = {'Error',txt};
legend([a1 a2],legendtxt,'FontSize',legend_sz,'interpreter','latex','
    location','southeast')

end

function plotPartC_Innovations(ezstore,szstore,svstore,xaxis_sz,yaxis_sz,
    legend_sz,params)
measx = 1:length(ezstore);
std_plot = 3; txt = [num2str(std_plot) '$\sigma$'];

a = params(1); k = params(2); b = params(3);
figure; grid on; set(gcf, 'WindowState', 'maximized'); hold on;
titleletxt = ['\textbf{Innovations versus Measurement Number using UFK with
    parameters:} '...

```

```

    '$\alpha$=',num2str(a),' $\kappa$=',num2str(k),' $\beta$=',num2str(b)
];
title(titletxt,'FontSize',25,'interpreter','latex')
a1 = plot(measx,ezstore(1,:),'-','Color','b','LineWidth',2,'MarkerSize'
,20);
b1 = plot(measx,std_plot*szstore(1,:),'-','Color','k','LineWidth',2,'
MarkerSize',20);
b2 = plot(measx,std_plot*svstore(1,:),'-','Color',[.7 .7 .7],'LineWidth'
,2,'MarkerSize',20);
plot(measx,-std_plot*szstore(1,:),'-','Color','k','LineWidth',2,'
MarkerSize',20);
plot(measx,-std_plot*svstore(1,:),'-','Color',[.7 .7 .7],'LineWidth',2,'
MarkerSize',20);
ylabel('State Innovation','FontSize',yaxis_sz,'interpreter','latex')
xlabel('Measurement number','FontSize',xaxis_sz,'interpreter','latex')
legendtxt = {'State Innov.', [txt ' Innov. cov.'],[txt ' Meas. noise cov
.']}];
legend([a1 b1 b2],legendtxt,'FontSize',legend_sz,'interpreter','latex','
location','southeast')
end

```

```

function plotPartC_EstimationError(exstore,sxstore,numPts,xaxis_sz,
yaxis_sz,legend_sz,params)
measx = 1:numPts;
std_plot = 3; txt = [num2str(std_plot) '$\sigma$'];

err_line_opts = {'-','LineWidth',1.3};
std_line_opts = {'-','LineWidth',1.3,'Color','k'};

measx1 = sort([measx measx]);
a = params(1); k = params(2); b = params(3);
figure; grid on; set(gcf, 'WindowState', 'maximized'); hold on;
titletxt = ['\textbf{State Estimation Error versus Measurement Number
using UFK with parameters:} '...
'$\alpha$=',num2str(a),' $\kappa$=',num2str(k),' $\beta$=',num2str(b
)];
title(titletxt,'FontSize',25,'interpreter','latex')
a1 = plot(measx1(3:end),exstore(1,2:end),err_line_opts{:});
a2 = plot(measx1(3:end),std_plot*sxstore(1,2:end),std_line_opts{:});
plot(measx1(3:end),-std_plot*sxstore(1,2:end),std_line_opts{:})
ylabel('State','FontSize',yaxis_sz,'interpreter','latex')
xlabel('Measurement number','FontSize',xaxis_sz,'interpreter','latex')
legendtxt = {'Error',txt};
legend([a1 a2],legendtxt,'FontSize',legend_sz,'interpreter','latex','
location','southeast')

end

```

```

function plotPartD_MD_state(MD_EKF,MD_UKF1,MD_UKF2,numPts,xaxis_sz,
yaxis_sz,legend_sz,params1,params2)

measx = 1:numPts;

```

```

line_opts1 = {'.-','LineWidth',2,'Color','k'};
line_opts2 = {'.-','LineWidth',2,'Color','b'};
line_opts3 = {'.-','LineWidth',2,'Color','r'};

a1 = params1(1); k1 = params1(2); b1 = params1(3);
a2 = params2(1); k2 = params2(2); b2 = params2(3);
figure; grid on; set(gcf, 'WindowState', 'maximized'); hold on;
titledtxt = '\textbf{Mahalanobis distance for each filter variation versus  
Measurement Number}';
UKFtxt1 = [' $\alpha$=',num2str(a1),' $\kappa$=',num2str(k1),' $\beta$=',  
num2str(b1)];
UKFtxt2 = [' $\alpha$=',num2str(a2),' $\kappa$=',num2str(k2),' $\beta$=',  
num2str(b2)];

title(titledtxt,'FontSize',25,'interpreter','latex')
% a1 = plot(measx,MD_EKF,line_opts1{:});
% a2 = plot(measx,MD_UKF1,line_opts2{:});
% a3 = plot(measx,MD_UKF2,line_opts3{:});
a1 = plot(measx,sqrt(MD_EKF),line_opts1{:});
a2 = plot(measx,sqrt(MD_UKF1),line_opts2{:});
a3 = plot(measx,sqrt(MD_UKF2),line_opts3{:});
ylabel('Mahalanobis distance (state)','FontSize',yaxis_sz,'interpreter','  
latex')
xlabel('Measurement number','FontSize',xaxis_sz,'interpreter','latex')
legendtxt = {'EKF',['UKF: ',UKFtxt1],['UKF: ',UKFtxt2]};
legend([a1 a2 a3],legendtxt,'FontSize',legend_sz,'interpreter','latex','  
location','southeast')

end

function plotPartE_EstimationErrorMC(exstore,sxstore,MC_variance,  
numTrials,numPts,xaxis_sz,yaxis_sz,legend_sz)
measx = 1:numPts;
std_plot = 3; txt = [num2str(std_plot) '$\sigma$'];

err_line_opts = {'-','LineWidth',1.3};
std_line_opts = {'-','LineWidth',1.3,'Color','k'};
std_line_opts2 = {'-','LineWidth',1.3,'Color',[.7 .7 .7]};

measx1 = sort([measx measx]);

figure; grid on; set(gcf, 'WindowState', 'maximized'); hold on;
title(['\textbf{State Estimation Error versus Measurement Number using  
EKF, Monte Carlo Analysis with: }',num2str(numTrials),'\textbf{ Trials  
}'],'FontSize',25,'interpreter','latex')
a1 = plot(measx1(3:end),exstore(1,2:end),err_line_opts{:});
a2 = plot(measx1(3:end),std_plot*sxstore(1,2:end),std_line_opts{:});
a3 = plot(measx1(3:end),std_plot*sqrt(MC_variance(1,2:end)),  
std_line_opts2{:});
plot(measx1(3:end),-std_plot*sxstore(1,2:end),std_line_opts{:})
plot(measx1(3:end),-std_plot*sqrt(MC_variance(1,2:end)),std_line_opts2  
{:})

```

```

ylabel('State','FontSize',yaxis_sz,'interpreter','latex')
xlabel('Measurement number','FontSize',xaxis_sz,'interpreter','latex')
legendtxt = {'Avg. Error', ['Avg. ',txt], ['MC state estimation error
    variance, ',txt]};
legend([a1 a2 a3],legendtxt,'FontSize',legend_sz,'interpreter','latex','
    location','southeast')

end

```