SOME STUDIES ON STATE ESTIMATION OF LINEAR AND NONLINEAR SYSTEMS

A Thesis

Submitted in partial fulfilment of the requirement for the Degree of

Master in Control System Engineering

(Electrical Engineering Department)

By

Prithwish Biswas

Registration No.: 163525 of 2022-2023

Examination Roll No.: M4CTL24006B

Under the Guidance of

Dr. Smita Sadhu (Ghosh)

Department of Electrical Engineering

Jadavpur University

Kolkata-700032

India

November, 2024

FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

CERTIFICATE

This is to certify that the dissertation entitled **"Some studies on state estimation of linear and nonlinear systems"** has been carried out by PRITHWISH BISWAS (University Registration No.: 163525 of 2022-2023) under my guidance and supervision and be accepted as partial fulfilment of the requirement for the Degree of Master in Control System Engineering.

Dr. Smita Sadhu (Ghosh)
Professor
Dept. of Electrical Engineering
Jadavpur University

Dr. Biswanath Roy
Head of the Department
Dept. of Electrical Engineering
Jadavpur University

Prof. Rajib Bandyopadhyay
Dean
Faculty of Engineering and Technology
Jadavpur University

FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

CERTIFICATE OF APPROVAL*

The forgoing thesis is hereby approved as a creditable study of an engineering subject and presented in a manner satisfactory to warrant acceptance as prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn there in but approve the thesis only for which it is submitted.

Committee on final examination for the evaluation of the thesis.

Signature of the Examiner

Signature of the Supervisor

*Only in the case the thesis is approved.

FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC THESIS

I hereby declare that this thesis, titled **"Some Studies on State Estimation of Linear and Nonlinear Systems"**, includes both a literature review and original research conducted by the undersigned as part of his Master's Degree in Control System Engineering. All information has been gathered and presented in compliance with academic standards and ethical guidelines. It is further affirmed that, in accordance with these standards, all sources and results not originating from this work have been duly cited and referenced.

Candidate Name: **PRITHWISH BISWAS**

Examination Roll. No.: M4CTL24006B

Thesis Title: **Some studies on state estimation of linear and nonlinear systems.**

Signature of the candidate

Acknowledgement

I would like to express my heartfelt gratitude to my thesis supervisor, Dr. Smita Sadhu (Ghosh), from the Department of Electrical Engineering at Jadavpur University, for granting me the opportunity to work under her guidance and encouraging me to delve into the field of Control System Engineering. Her patient mentoring, insightful feedback, and unwavering support have been instrumental in shaping my research.

I specially thank Prof. Tapan Kumar Ghoshal, Emeritus Professor, Electrical Engineering Department, Jadavpur University, Kolkata for his innovative discussions and for sharing his valuable suggestions, ideas and thoughts, which inspired me to do a project in this domain as well as helped me throughout my thesis work.

I would also like to thank my senior Mrs. Ankita Muhury, PhD scholar from the Electrical Engineering Department, Jadavpur University for her guidance in this project. It was a wonderful experience working alongside her. Also, I would like to thank my classmates for their help and support.

Finally, I extend my sincere thanks to my parents, Mr. Palas Biswas and Mrs. Sachi Biswas, for their constant support, encouragement, and steadfast belief in my abilities.

Date:

Place:

PRITHWISH BISWAS
Department of Electrical Engineering
Examination Roll No. : M4CTL24006B
Jadavpur University

Abstract

This thesis investigates the performance of linear and nonlinear estimation methods, focusing on Kalman Filter (KF), Extended Kalman Filter (EKF), and Adaptive Kalman Filter (AKF) in estimating dynamic system states. A comprehensive MATLAB implementation is developed for 1D, 2D, 3D, and N-Dimensional Kalman Filters, based on established algorithms, to analyze and validate these filters' performance across a range of linear and nonlinear systems. This work provides a comparative study between the EKF and the recently proposed AKF in nonlinear environments, with an emphasis on their responses to measurement noise covariance.

The study primarily evaluates the filter's ability to minimize estimation error under varying conditions of measurement noise covariance, R. It is observed that the EKF demonstrates robust performance in systems with mild nonlinearities, given that the measurement noise covariance is accurately known or estimated. However, in scenarios where measurement noise covariance is uncertain or unknown, the AKF exhibits superior adaptability and precision, thus outperforming the EKF. This adaptability of the AKF makes it a preferable choice for applications where measurement noise characteristics cannot be precisely determined.

Results indicate that when the actual measurement noise covariance is known or reasonably approximated, both AKF and EKF offer comparable accuracy. Nevertheless, when covariance knowledge is limited, AKF's performance advantage becomes significant. While the MATLAB code developed for this study successfully supports these analyses, further optimization could reduce computational demands, particularly for extensive Monte Carlo simulations. Additionally, refining the AKF algorithm could enhance its applicability to systems with stronger nonlinear behaviors, potentially expanding its practical usage across more complex, real-world systems.

In conclusion, this thesis underscores the importance of adaptive filtering in scenarios of measurement uncertainty and highlights potential areas for advancing Kalman Filter implementations in both theoretical and applied contexts.

Table Of Contents

Chapter 1 - Introduction

The Kalman filter was created by Rudolf E. Kalman in the 1960s, an American-Hungarian electrical engineer and mathematician. The method was originally introduced as a way to predict a system's future state based on a series of measurements that contain noise. Kalman's work revolutionized the field of control systems engineering, particularly with its application in the Apollo space program, where it was employed to navigate the lunar module. The main concept of the filter is to offer a recursive method for estimating the internal state of a process that cannot be directly observed, using noisy sensor data for the calculations [Brown2012].

The Kalman filter is a robust mathematical technique commonly used in control systems, signal processing, and estimation theory. Since its inception, it has gained widespread application in diverse areas such as robotics, aerospace engineering, navigation, and finance. This thesis aims to provide a comprehensive overview of the Kalman filter, tracing its historical development, exploring its types, and highlighting its strengths and limitations. Additionally, this thesis discusses the application of the Kalman filter in estimating the states of both linear and nonlinear systems, a critical task in many control systems [Brown2012].

There are various usage of the Kalman filter in several engineering domains apart from the domain of electrical engineering. In control systems, precise state estimation is essential for achieving effective feedback control. The Kalman filter excels in estimating the true state of a system in real time by using a combination of noisy sensor data and a model of the system's dynamics. In

various engineering domains the Kalman filter is used extensively for both linear and non-linear models such as -

- **Navigation Systems**: In applications like GPS or inertial navigation, where the motion of a vehicle or object can be represented using linear equations, the Kalman filter is ideal for estimating the position and velocity of the vehicle in real time.
- **Control Systems**: In automatic control systems, such as those used in aircraft or industrial processes, the Kalman filter helps estimate the internal states of the system based on sensor measurements, enabling effective feedback control.
- **Signal Processing**: The Kalman filter is used to estimate signals that are corrupted by noise, improving the clarity of the signal in communication systems or audio processing.
- **SOC Battery Estimation**: Electric vehicle batteries are inherently nonlinear systems, with their performance influenced by factors such as temperature, current, and voltage. Estimating the State of Charge (SOC) [Proctor2020] , [Ilies2020] involves monitoring the remaining energy in the battery as it depletes or charges over time.
- **Robotics**: In autonomous robots, the motion and sensor models are often nonlinear, especially when dealing with rotation or complex environments. The EKF and UKF are used for simultaneous localization and mapping (SLAM), where the robot must estimate both its position and the environment it is navigating through.
- **Aerospace**: The trajectory of a spacecraft, particularly during re-entry or in complex manoeuvres, is highly nonlinear. The EKF and UKF are

employed to estimate the spacecraft's position and velocity for navigation and control [Das2014].

Over the years, several variations of the Kalman filter have been developed to address specific challenges. The standard Kalman filter is designed for systems that can be modelled using linear dynamics and Gaussian noise. However, many real-world systems are nonlinear, prompting the development of other variants

1. **Kalman Filter (KF)**: Used for systems with linear dynamics. It offers an optimal estimation approach when both the system and the noise adhere to Gaussian distributions. KF is computationally simple but cannot handle nonlinear systems effectively.
2. **Extended Kalman Filter (EKF)**: This filter is used for managing nonlinear systems by simplifying their complexity. It does this by linearizing the system around the current state estimate, using Jacobian to approximate the nonlinear behaviour. EKF is widely used for battery state estimation but can be less accurate for systems with strong nonlinearities [Shrivastava2019].
3. **Adaptive Extended Kalman Filter (AEKF)**: An enhancement of the EKF, the AEKF adjusts the process and measurement noise covariances during the estimation process. This allows the filter to better handle varying noise levels and system conditions, improving accuracy and reducing divergence [Shrivastava2019].
4. **Unscented Kalman Filter (UKF)**: This filter is more accurate for **nonlinear systems** as it avoids linearization. Instead, it uses a set of sample points (sigma points) to more precisely estimate the state of a system. The UKF is typically more accurate than the EKF in highly nonlinear situations [Shrivastava2019].

5. **Adaptive Unscented Kalman Filter (AUKF)**: Like the UKF, but with the added capability to adaptively adjust the noise covariance matrices. It is very useful for nonlinear systems having non-additive measurement noise with the unknown noise statistics. This makes the AUKF particularly useful in real-time applications with changing system dynamics and noise levels [Das2015].
6. **Square-root Unscented Kalman Filter (SR-UKF)**: This variant reduces the computational burden of the UKF by using square-root decompositions, making it more efficient for systems where computational resources are limited [Shrivastava2019].
7. **Central Difference Kalman Filter (CDKF)**: Uses a simpler method to approximate derivatives, reducing computational complexity compared to the UKF. It is useful for applications requiring efficient computation without significant loss in accuracy [Shrivastava2019].
8. **Cubature Kalman Filter (CKF)**: This filter uses a cubature integration method to handle high-dimensional nonlinear systems more effectively than the UKF. It offers better performance for certain complex systems but requires more computational resources [Shrivastava2019].

In the case of linear systems, the standard Kalman filter is an optimal solution. Linear systems can be described by state-space equations, where the future state of the system depends linearly on the current state and control inputs. The Kalman filter works by using a predictive step, where the future state is estimated based on a mathematical model of the system, and a corrective step, where the prediction is updated using noisy measurements. This two-step process allows the Kalman filter to continuously refine the estimate of the system's state in real time.

Most real-world systems are nonlinear, which complicates state estimation. The Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) are the two most commonly used variations for dealing with nonlinearities. The EKF approximates the nonlinear dynamics by linearizing them around the current state estimate, but this approximation can introduce errors in systems with strong nonlinear behaviour. On the other hand, the UKF uses a more sophisticated approach, approximating the state distribution with a set of weighted points that are propagated through the nonlinear system. While more computationally expensive, the UKF generally provides better estimates for highly nonlinear systems.

However in this thesis we are not going to study about all the types of Kalman filters present. We will be discussing mainly the Kalman filter (KF), Extended Kalman filter (EKF) and Adaptive Kalman filter (AKF) and will do a comparative study on how these filters are performing in estimating the states of linear and non-linear systems.

Chapter 2 - Literature survey

Introduction

Kalman filters are a cornerstone in the field of estimation theory, with applications spanning various domains, including aerospace, robotics, control systems, and battery management. The Kalman filter is fundamentally a recursive algorithm that provides estimates of the internal states of a system given noisy observations. Over the decades, several variants of the Kalman filter have been developed to address the limitations of the standard Kalman filter in nonlinear, non-Gaussian, and adaptive environments. This literature survey reviews the key Kalman filter variants, focusing on the Standard Kalman Filter (KF), Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF), and Adaptive Kalman Filter (AKF).

Kalman Filter (KF)

The Kalman Filter algorithm operates in two main steps: prediction and update. During the prediction step, the filter projects the current state estimate forward in time to obtain the predicted state and its uncertainty. In the update step, the filter corrects this prediction using the latest measurement. The KF is known for being computationally efficient, making it suitable for real-time applications. However, its effectiveness is restricted to linear systems, and its performance degrades significantly in nonlinear or non-Gaussian settings. Despite these limitations, the KF remains a fundamental tool, particularly in systems where the assumptions of linearity and Gaussian noise hold true.

Extended Kalman Filter (EKF)

The Extended Kalman Filter (EKF) was developed to extend the application of the Kalman filter to nonlinear systems. The EKF approximates the nonlinear system by linearizing the system dynamics around the current state estimate. This is achieved using a first-order Taylor series expansion, which allows the EKF to apply the standard Kalman filtering process to the linearized system. While this method enables the EKF to handle a broader range of applications, it introduces new challenges. The linearization process can lead to inaccuracies, especially in highly nonlinear systems where higher-order terms of the Taylor series become significant. Moreover, the EKF assumes that the noise in the system remains Gaussian, which is often not the case in real-world applications. Despite these limitations, the EKF is widely used in various fields, including navigation systems, robotics, and battery management systems, due to its ability to manage nonlinear dynamics to some extent [Takayama2024].

Unscented Kalman Filter (UKF)

The Unscented Kalman Filter (UKF) was developed as a more robust alternative to the EKF, specifically designed to handle the shortcomings of linearization. Instead of linearizing the system dynamics, the UKF employs the Unscented Transform, a deterministic sampling technique that selects a set of points (called sigma points) around the mean of the state estimate. These points are then propagated through the nonlinear system equations, allowing the UKF to capture the mean and covariance of the state distribution more accurately than the EKF. This method results in better performance, particularly in systems with significant nonlinearities. The UKF has been shown to provide more accurate estimates than the EKF in many practical applications, including

vehicle tracking, spacecraft navigation, and sensor fusion in autonomous systems. However, the UKF is computationally more intensive than the EKF, which can be a limiting factor in real-time applications [Shrivastava2019].

Adaptive Kalman Filter (AKF)

The Adaptive Kalman Filter (AKF) represents a class of filters that address one of the key limitations of the standard and extended Kalman filters the need for accurate knowledge of process and measurement noise covariances. In many practical situations, these noise characteristics are either unknown or vary over time. The AKF dynamically adjusts its parameters, such as the process noise covariance (Q) and measurement noise covariance (R), based on observed data. This adaptation is crucial for maintaining filter performance in non-stationary environments, where the noise characteristics change over time due to varying operating conditions. For instance, in battery management systems, the AKF can adapt to changes in battery behaviour caused by temperature fluctuations, aging, or varying load profiles. By continuously updating its parameters, the AKF enhances the accuracy and robustness of state estimation, reducing the likelihood of filter divergence, a situation where the filter's estimates become increasingly inaccurate over time [Takayama2024].

Comparative Analysis of Kalman Filter Variants

The choice of Kalman filter variant depends heavily on the specific application and the nature of the system being modelled. The Standard Kalman Filter is optimal for linear systems with well-characterized Gaussian noise, offering simplicity and efficiency. However, its performance degrades in nonlinear or non-Gaussian settings, necessitating the use of more advanced filters like the EKF or UKF[Shrivastava2019].

The EKF is suitable for systems with mild nonlinearities, where the first-order approximation provided by linearization is sufficient. However, in systems with significant nonlinearities or where a more accurate estimation of the state distribution is required, the UKF is often the better choice due to its ability to propagate the mean and covariance through nonlinear transformations more accurately [Shrivastava2019].

The AKF is particularly beneficial in applications where noise characteristics are time-varying or not well known. Its ability to adapt to changing conditions makes it ideal for non-stationary environments, although this comes with increased computational complexity. For example, in dynamic environments such as autonomous vehicles, the AKF can significantly improve estimation accuracy by adjusting its parameters in real-time as the vehicle encounters different terrains or sensor conditions.

In summary, while the Standard Kalman Filter is optimal for linear, Gaussian systems, the EKF, UKF, and AKF each offer specific advantages for handling nonlinearities and time-varying noise, albeit at the cost of increased computational requirements.

Applications in Engineering and Technology

Kalman Filters are extensively used across various engineering disciplines due to their versatility in state estimation. In battery management systems (BMS) [Ilies2020], Kalman Filters are crucial for State of Charge (SOC) estimation, a critical parameter for ensuring the safe and efficient operation of batteries [Shrivastava2019]. The EKF and UKF are commonly employed in this context due to their ability to handle the nonlinear dynamics of battery models. For instance, the nonlinear double-capacitor model of a battery, which is often used to represent the complex electrochemical processes, is better suited to the EKF

and UKF than the standard KF[Proctor2020]. The AKF, with its adaptive capabilities, is particularly useful in BMS applications where the battery's operating conditions change over time, such as variations in temperature or load profiles.

In the field of autonomous vehicles and robotics, Kalman Filters are integral to sensor fusion, where data from multiple sensors, such as GPS, IMUs (Inertial Measurement Units), and cameras, are combined to estimate the vehicle's position and velocity. The UKF is often preferred in these applications due to its superior handling of nonlinear sensor models. Additionally, the AKF is employed to adjust the filter parameters in real-time, ensuring robust state estimation even when the vehicle operates in dynamic environments with varying sensor noise levels.

Kalman Filters are also widely used in aerospace applications, particularly in navigation and control systems for aircraft and spacecraft. The EKF and UKF are used for trajectory estimation, attitude determination, and orbit determination, where the nonlinearities of the system dynamics are significant. The adaptability of the AKF is beneficial in these high-precision applications, where the noise characteristics can change due to varying atmospheric conditions or different phases of flight [Das2014].

Challenges and Future Directions

Despite the widespread adoption of Kalman Filters in various applications, several challenges remain. One of the most significant challenges is the assumption of Gaussian noise, which may not hold in many real-world scenarios. Non-Gaussian noise can lead to suboptimal performance and even filter divergence. To address this, ongoing research is exploring more robust

filtering techniques, such as particle filters, which do not assume Gaussian noise and can provide better performance in such scenarios.

Another challenge is the computational complexity associated with the UKF and AKF. While these filters offer improved accuracy over the standard KF and EKF, they require more computational resources, which can be a limiting factor in real-time applications or systems with limited processing power. Future research is likely to focus on developing more efficient algorithms that can offer the benefits of the UKF and AKF while reducing computational overhead.

The integration of Kalman Filters with machine learning techniques is also an emerging area of research. For instance, combining Kalman Filters with neural networks could enhance their ability to model complex, nonlinear systems without relying on explicit system models. This approach could lead to more accurate and robust state estimation in a broader range of applications.

Conclusion

Kalman Filters, particularly the Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF), and Adaptive Kalman Filter (AKF), have become essential tools in modern engineering, enabling precise state estimation in the presence of noise and uncertainty. Each variant offers unique advantages depending on the nature of the system and the noise environment. While the standard Kalman Filter is optimal for linear, Gaussian systems, the EKF and UKF extend its applicability to nonlinear systems, and the AKF provides the adaptability needed in time-varying environments. As research continues, we can expect further advancements in the accuracy, robustness, and computational efficiency of these filters, expanding their applicability to even more challenging and dynamic systems.

Chapter 3 – Brief overview of the Kalman Filter (KF)

3.0 Introduction

The Kalman Filter is a recursive algorithm used to estimate the state of a system from noisy measurements. It minimizes the mean square error between the estimated and true states of the system by combining past information and new measurements in an optimal way. Let us understand with a simple example, imagine you're trying to track the position of a car using a GPS device. The GPS provides noisy data due to interference, weather, etc., so the position estimates aren't perfect. The Kalman filter helps by using both the previous position of the car and a model of how fast the car is moving to predict where it should be. Then, it updates this prediction using the GPS reading, while also accounting for the noise. Over time, this process produces a very accurate estimate of where the car actually is, despite the noisy GPS measurements. Also, we have to keep it in mind that the Kalman filter works only in discrete domain as it checks for the updates on the particular time instance and accordingly proceeds. In this scenario firstly we will be requiring the plant's the state space input and output equations are represented in the discrete domain and then we can move on with its working. In summary, the Kalman filter is a method that combines predictions from a system model with noisy measurements to provide an accurate estimate of the system's state. It's powerful because it can continuously update its estimate as new data comes in, making it ideal for tracking and estimation in dynamic systems. It's widely applied in fields like

navigation, control systems, robotics, and economics, where you need to make sense of noisy data to predict or track the state of a system [Brown2012].

3.1 Working principle

To explain how it works, let's break it down step by step -

a) **Basic idea:**

The Kalman filter continuously estimates the current state of a system by combining two pieces of information [Brown2012]:

- **Predicted state**: What the system should be, based on a model of how the system behaves.
- **Measured state**: What the system seems to be, based on sensor readings or observations, which often contain noise or inaccuracies.

By blending these two inputs, the Kalman filter finds a best estimate that minimizes error.

b) **How it works**:

The process of the Kalman filter can be understood as happening in two main steps, which repeat at every time step.

Step 1: Prediction (What should happen)*:*

The Kalman filter uses the previous state (the last known information) and a model of the system (how things are expected to evolve) to predict what the current state should be. This is based on knowledge of how the system behaves

over time. However, this prediction is only an estimate, and it might not be entirely accurate [Brown2012].

Step 2: Update (What actually happens)*:*

Once a measurement or observation of the current state is made (for example, from a sensor), the Kalman filter compares this measured state to its predicted state. Since measurements can have noise (random errors), the filter doesn't trust the measurement completely. Instead, it blends the prediction and the measurement to get a new, updated estimate that is more reliable than either one on its own [Brown2012].

c) **Balancing prediction and measurement**:

The key strength of the Kalman filter is its ability to weigh the predicted state and the measured state intelligently. If the measurement seems very noisy (unreliable), the filter will trust the prediction more. If the prediction model is not very precise, the filter will lean more on the actual measurement. This balance is achieved by assigning "weights" based on how confident the filter is about the prediction versus the measurement [Brown2012].

d) **Continuous loop**:

This process of prediction and update happens at each time step:

- First, the filter makes a prediction based on the past state and system model.
- Then, it takes in the new measurement, combines it with the prediction, and updates its estimate.

This allows the filter to track the state of the system over time, even if the data is noisy or incomplete [Brown2012].

e) **Why it's useful**:

- **Noise Handling**: The Kalman filter is very good at filtering out random noise in measurements and providing a smoother, more accurate estimate of the system's state.
- **Real-Time Updates**: It works recursively, meaning it updates estimates as soon as new data is available, making it suitable for real-time applications like GPS tracking or robotic navigation.
- **Uncertainty Management**: It can handle uncertainty in both the system model and the measurements, which makes it versatile for many applications.

3.2 One-dimensional Kalman filter

The one-dimensional Kalman filter is the simplest form of the filter, used when the system can be described with a single state variable and the measurements are scalar values. In this case, the state of the system at time k is represented by a scalar x_k, and the filter estimates this state using noisy measurements z_k. Since in this case all the state variables are scalar in nature thus these are nothing but matrices with a single value. So only for this case we are not going to use matrices rather just simple variables storing single values in them. The filter follows two key steps prediction and update. Before that we need to come up with the discrete time domain input and output equations of the given plant. Here it is represented as follows [Brown2012]

$$x_{k+1} = F_k x_k + w_k \dots\dots\dots (1)$$

$$y_k = H_k x_k + v_k \dots\dots\dots (2)$$

Where:

- x_{k+1} is the predicted state.
- F_k is the state transition matrix, describing how the state evolves from time step k to $k+1$.
- x_k is the state vector at time t_k.
- w_k is the process noise (Gaussian white noise).
- y_k is the output state.
- H_k is the observation matrix.
- v_k is the measurement noise (Gaussian white noise).

1. Prediction step

The filter predicts the next state of the system using a model of the system's dynamics. It also projects the error covariance matrix, which quantifies the uncertainty in the prediction. The state prediction can be described by [Brown2012]:

$$\hat{x}_{k+1}^{-} = F_k \hat{x}_k \dots\dots\dots (3)$$

Where:

- $\hat{x}_{k+1}^{-}$ is the predicted state.
- F_k is the state transition matrix, describing how the state evolves from time step k to $k+1$.
- x_k is the state vector at time t_k.

The error covariance matrix, P_k, is updated as [Brown2012]:

$$P_{k+1}^{-} = F_k P_k F_k^T + Q_k \dots\dots\dots (4)$$

Where:

- P_{k+1}^{-} is the priori predicted error covariance.
- F_k is the state transition matrix, describing how the state evolves from time step k to $k+1$.
- P_k is the error covariance matrix associated with the estimate $\hat{x}_k$
- Q_k is the covariance matrix of the process noise w_k.
- w_k is the process noise (Gaussian white noise).

2. Determination of Kalman gain

In this step, the Kalman gain is found K_k, which determines how much the prediction should be corrected based on the new measurement [Brown2012]:

$$K_k = P_{k+1}^{-} H_k^T (H_k P_{k+1}^{-} H_k^T + R_k)^{-1} \dots\dots\dots (5)$$

Where:

- H_k is the observation matrix.
- K_k is the Kalman gain, a blending factor for updating the estimate using measurements.
- P_{k+1}^{-} is the priori predicted error covariance.
- $R_k$is the covariance matrix of the measurement noise v_k.

3. Update step

In this step many variables are updated subsequently such as $\hat{x}_{k+1}$, P_{k+1} , e_k also the filter incorporates the new measurement z_k to correct the prediction.

The state estimate is then updated [Brown2012]:

$$\hat{x}_{k+1} = \hat{x}_{k+1}^{-} + K_k(z_k - H_k\hat{x}_{k+1}^{-}) \dots\dots\dots (6)$$

Where:

- $\hat{x}_{k+1}$ is the posterior predicted state.
- $\hat{x}_{k+1}^{-}$ is the priori predicted state.
- K_k is the Kalman gain, a blending factor for updating the estimate using measurements.
- z_k is the observation (measurement) vector at time t_k.
- H_k is the observation matrix, mapping the true state to the observed state.

The error covariance matrix is updated to reflect the improved estimate [Brown2012]:

$$P_{k+1} = (I - K_k H_k)P_{k+1}^{-} \dots\dots\dots (7)$$

Where:

- P_{k+1} is the predicted error covariance.
- K_k is the Kalman gain, a blending factor for updating the estimate using measurements.
- H_k is the observation matrix, mapping the true state to the observed state.
- P_{k+1}^{-} is the priori predicted error covariance.

Also we have to find the error matrix to actually find how good the filter is working, so it can be found out as follows [Brown2012]

$$e_k = x_k - \hat{x}_k \dots\dots\dots (8)$$

Where:

- e_k is the error matrix.
- x_k is the actual value of the state.
- $\hat{x}_k$ is the estimated value of the state.

4. Kalman filter algorithm

- **Initialization**: Start with an initial estimate $\hat{x}_0$ and covariance P_0.
- **Prediction**:
  - Predict the next state $\hat{x}_{k+1}^-$.
  - Update the error covariance P_{k+1}^-.
- **Measurement**: Obtain the new measurement z_k.
- **Update**:
  - Compute the Kalman gain K_k.
  - Update the state estimate $\hat{x}_{k+1}$.
  - Update the error covariance P_{k+1}.
- **Repeat** for each new measurement.

5. Flowchart

We have already discussed about the working algorithm and the steps of the Kalman filter. Now with the help of a flowchart we will understand how actually the steps has been followed and implemented in the MATLAB. This portion of the flowchart will be in a loop since it is a recursive process and requires a number of iterations to get better results [Brown2012].

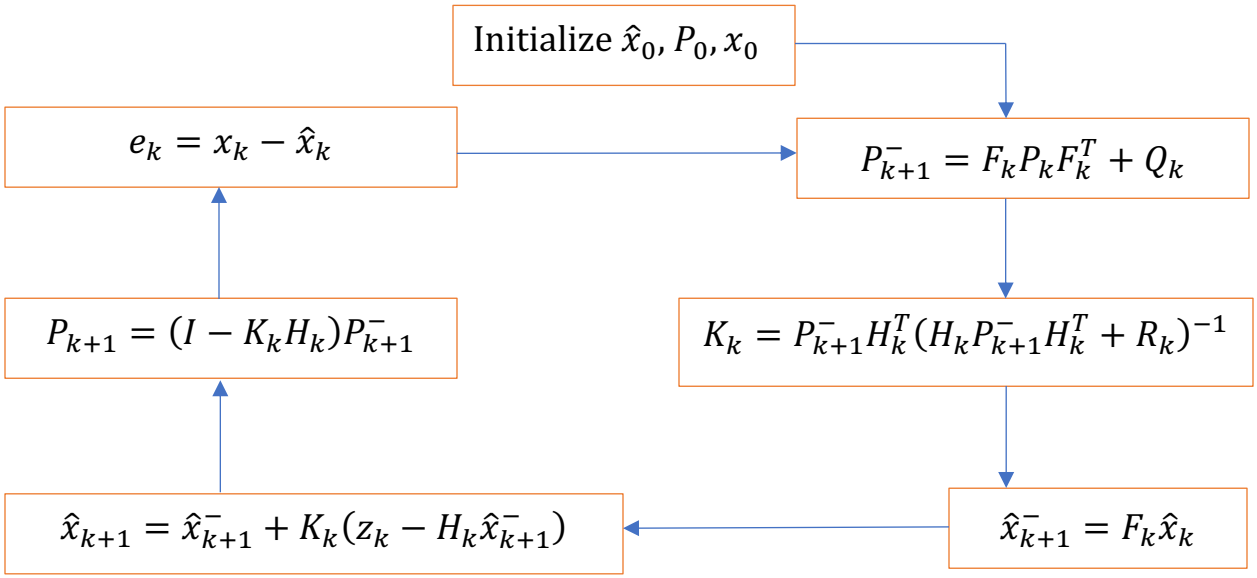

Fig.1 Flowchart of the Kalman Filter

This filter optimally balances the information from the predicted model and the noisy measurements, providing an accurate state estimate even in uncertain conditions. This overview summarizes the basic principles and operation of the discrete Kalman filter, with its key mathematical equations, steps, and a flowchart for understanding it's working.

3.3 Two-dimensional Kalman filter

The two-dimensional Kalman filter extends the basic concept to systems with two state variables. A two-dimensional Kalman filter is an extension of the Kalman filter designed to estimate the state of a system where there are two variables or dimensions involved, such as position in 2D space (e.g., x and y coordinates). Like the one-dimensional version, it works by combining predictions based on a system model with noisy measurements to produce more accurate estimates over time. These could represent quantities such as position and velocity in two-dimensional space. Let's understand with a simple example imagine you're tracking the position of a moving object, such as a robot, on a 2D grid (like a plane). The robot moves around, and you want to estimate its position (x and y coordinates) over time based on sensor readings. The sensor readings are noisy, so the measurements might not be fully reliable. The two-dimensional Kalman filter helps by predicting the next (x, y) position of the robot based on its previous position and speed. Also updating this prediction with new sensor data about the robot's current position, while taking into account the noise in the measurements.

The entire algorithm and the steps that we had followed in one dimensional Kalman Filter are exactly same as in two dimensional Kalman filter, the only difference here is that the variable in one dimensional Kalman filter were singleton matrices but in two dimensional it will be not. We will see here how the dimensions of each of the variables changes here.

The state vector is now a column matrix:

$$\boldsymbol{x}_k = \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} \dots \dots \dots (9)$$

Where $x_{1,k}$ is the first state, and $x_{2,k}$ is the second state.

The filter here also follows two key steps **prediction** and **update**. Before that we need to come up with the discrete time domain input and output equations of the given plant. Here it is represented as follows -

$$\boldsymbol{x}_{k+1} = \boldsymbol{F}_k \boldsymbol{x}_k + \boldsymbol{w}_k \dots\dots\dots (1.1)$$

$$\boldsymbol{y}_k = \boldsymbol{H}_k \boldsymbol{x}_k + \boldsymbol{v}_k \dots\dots\dots (2.1)$$

Where:

- $\boldsymbol{x}_{k+1}$ is the predicted state matrix of (2×1).
- $\boldsymbol{F}_k$ is the state transition matrix of (2×2), describing how the state evolves from time step k to $k + 1$.
- $\boldsymbol{x}_k$ is the state vector of (2×1) at time t_k.
- $\boldsymbol{w}_k$ is the process noise (Gaussian white noise) vector of (2×1).
- $\boldsymbol{y}_k$ is the output state of (1×1).
- $\boldsymbol{H}_k$ is the observation matrix of (1×2).
- $\boldsymbol{v}_k$ is the measurement noise (Gaussian white noise) matrix of (1×1).

1. Prediction step

The filter predicts the next state of the system using a model of the system's dynamics. It also projects the error covariance matrix, which quantifies the uncertainty in the prediction. The state prediction can be described by:

$$\hat{\boldsymbol{x}}_{k+1}^{-} = \boldsymbol{F}_k \hat{\boldsymbol{x}}_k \dots\dots\dots (3.1)$$

Where:

- $\widehat{\boldsymbol{x}}_{k+1}^{-}$ is the predicted state vector of (2×1).
- $\boldsymbol{F}_k$ is the state transition matrix of (2×2), describing how the state evolves from time step k to $k+1$.
- $\boldsymbol{x}_k$ is the state vector of (2×1) at time t_k.

The error covariance matrix, $\boldsymbol{P}_k$, is updated as:

$$\boldsymbol{P}_{k+1}^{-} = \boldsymbol{F}_k \boldsymbol{P}_k \boldsymbol{F}_k^T + \boldsymbol{Q}_k \ldots\ldots\ldots (4.1)$$

Where:

- $\boldsymbol{P}_{k+1}^{-}$ is the priori predicted error covariance matrix of (2×2).
- $\boldsymbol{F}_k$ is the state transition matrix of (2×2), describing how the state evolves from time step k to $k+1$.
- $\boldsymbol{P}_k$ is the error covariance matrix of (2×2) associated with the estimate $\widehat{\boldsymbol{x}}_k$.
- $\boldsymbol{Q}_k$ is the covariance matrix of (2×2) the process noise $\boldsymbol{w}_k$.
- $\boldsymbol{w}_k$ is the process noise (Gaussian white noise) vector of (2×1).

2. Determination of Kalman gain

In this step, the Kalman gain is found K_k, which determines how much the prediction should be corrected based on the new measurement:

$$\boldsymbol{K}_k = \boldsymbol{P}_{k+1}^{-} \boldsymbol{H}_k^T (\boldsymbol{H}_k \boldsymbol{P}_{k+1}^{-} \boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1} \ldots\ldots\ldots (5.1)$$

Where:

- $\boldsymbol{H}_k$ is the observation matrix of (2×1).
- $\boldsymbol{K}_k$ is the Kalman gain matrix of (2×2), a blending factor for updating the estimate using measurements.
- $\boldsymbol{P}_{k+1}^{-}$ is the priori predicted error covariance matrix of (2×2).
- $\boldsymbol{R}_k$is the covariance matrix of (1×1) of the measurement noise $\boldsymbol{v}_k$.

3. Update step

In this step many variables are updated subsequently such as $\widehat{\boldsymbol{x}}_{k+1}$, $\boldsymbol{P}_{k+1}$, $\boldsymbol{e}_k$ also the filter incorporates the new measurement z_k to correct the prediction.

The state estimate is then updated:

$$\widehat{\boldsymbol{x}}_{k+1} = \widehat{\boldsymbol{x}}_{k+1}^{-} + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{H}_k\widehat{\boldsymbol{x}}_{k+1}^{-}) \dots\dots\dots (6.1)$$

Where:

- $\widehat{\boldsymbol{x}}_{k+1}$ is the posterior predicted state matrix of (2×1).
- $\widehat{\boldsymbol{x}}_{k+1}^{-}$ is the priori predicted state matrix of (2×1).
- $\boldsymbol{K}_k$ is the Kalman gain matrix of (2×2), a blending factor for updating the estimate using measurements.
- $\boldsymbol{z}_k$ is the observation (measurement) vector of (1×1)at time t_k.
- $\boldsymbol{H}_k$ is the observation matrix of (1×2), mapping the true state to the observed state.

The error covariance matrix is updated to reflect the improved estimate:

$$\boldsymbol{P}_{k+1} = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{P}_{k+1}^{-} \dots\dots\dots (7.1)$$

Where:

- $\boldsymbol{P}_{k+1}$ is the predicted error covariance matrix of (2×2).
- $\boldsymbol{K}_k$is the Kalman gain matrix of (2×2), a blending factor for updating the estimate using measurements.
- $\boldsymbol{H}_k$ is the observation matrix of (1×2), mapping the true state to the observed state.
- $\boldsymbol{P}_{k+1}^{-}$ is the priori predicted error covariance matrix of (2×2).

Also we have to find the error matrix to actually find how good the filter is working, so it can be found out by

$$\boldsymbol{e}_k = \boldsymbol{x}_k - \widehat{\boldsymbol{x}}_k \ldots\ldots\ldots (8.1)$$

Where:

- $\boldsymbol{e}_k$ is the error matrix of (2×1).
- $\boldsymbol{x}_k$ is the actual value of the state vector of (2×1).
- $\widehat{\boldsymbol{x}}_k$ is the estimated value of the state vector of (2×1).

4. Kalman filter algorithm

The algorithm works exactly in the same way as already been shown in sub-part (4) of section (3.2).

5. Flowchart

The flowchart is also the same as shown in Fig. (1).

3.4 Three-dimensional Kalman filter

A three-dimensional Kalman filter is an extension of the Kalman filter that estimates the state of a system involving three variables or dimensions. These dimensions are typically related to physical quantities such as position (x, y, z), velocity, or other attributes that change in 3D space. The Kalman filter operates by combining model-based predictions with noisy measurements to refine the estimate of the system's state over time. Let us understand with a simple example imagine you are tracking the 3D position of a drone in flight, where its position is defined by the x, y, and z coordinates (representing latitude, longitude, and altitude). As the drone moves, GPS sensors provide noisy data about its position in 3D space. The three-dimensional Kalman filter helps by:

- Predicting the drone's next position (x, y, z) based on its previous position and speed.
- Updating this prediction with new, possibly noisy GPS measurements of the drone's current position, resulting in a more accurate estimate.

With each new set of measurements, the Kalman filter adjusts its estimates, providing smooth and accurate tracking of the drone's 3D position over time.

In this case the state vector becomes:

$$\boldsymbol{x}_k = \begin{bmatrix} x_{1,k} \\ x_{2,k} \\ x_{3,k} \end{bmatrix} \ldots\ldots\ldots (10)$$

Where $x_{1,k}$ is the first state, $x_{2,k}$ is the second state and $x_{3,k}$ is the third state.

Here also the filter follows two key steps prediction and update. Before that we again need to come up with the discrete time domain input and output equations of the given plant. Here it is represented as follows -

$$\boldsymbol{x}_{k+1} = \boldsymbol{F}_k \boldsymbol{x}_k + \boldsymbol{w}_k \ldots\ldots\ldots (1.2)$$

$$\boldsymbol{y}_k = \boldsymbol{H}_k \boldsymbol{x}_k + \boldsymbol{v}_k \ldots\ldots\ldots (2.2)$$

Where:

- $\boldsymbol{x}_{k+1}$ is the predicted state matrix of (3×1).
- $\boldsymbol{F}_k$ is the state transition matrix of (3×3), describing how the state evolves from time step k to $k + 1$.
- $\boldsymbol{x}_k$ is the state vector of (3×1) at time t_k.
- $\boldsymbol{w}_k$ is the process noise (Gaussian white noise) vector of (3×1).
- $\boldsymbol{y}_k$ is the output state of (1×1).
- $\boldsymbol{H}_k$ is the observation matrix of (1×3).
- $\boldsymbol{v}_k$ is the measurement noise (Gaussian white noise) matrix of (1×1).

1. Prediction step

The filter predicts the next state of the system using a model of the system's dynamics. It also projects the error covariance matrix, which quantifies the uncertainty in the prediction. The state prediction can be described by:

$$\hat{\boldsymbol{x}}_{k+1}^{-} = \boldsymbol{F}_k \hat{\boldsymbol{x}}_k \ldots\ldots\ldots (3.2)$$

Where:

- $\widehat{\boldsymbol{x}}_{k+1}^{-}$ is the predicted state vector of (3×1).
- $\boldsymbol{F}_k$ is the state transition matrix of (3×3), describing how the state evolves from time step k to $k+1$.
- $\boldsymbol{x}_k$ is the state vector of (3×1) at time t_k.

The error covariance matrix, $\boldsymbol{P}_k$, is updated as:

$$\boldsymbol{P}_{k+1}^{-} = \boldsymbol{F}_k \boldsymbol{P}_k \boldsymbol{F}_k^T + \boldsymbol{Q}_k \ldots\ldots\ldots (4.2)$$

Where:

- $\boldsymbol{P}_{k+1}^{-}$ is the priori predicted error covariance matrix of (3×3).
- $\boldsymbol{F}_k$ is the state transition matrix of (3×3), describing how the state evolves from time step k to $k+1$.
- $\boldsymbol{P}_k$ is the error covariance matrix of (3×3) associated with the estimate $\widehat{\boldsymbol{x}}_k$.
- $\boldsymbol{Q}_k$ is the covariance matrix of (3×3) the process noise $\boldsymbol{w}_k$.
- $\boldsymbol{w}_k$ is the process noise (Gaussian white noise) vector of (3×1).

2. Determination of Kalman gain

In this step, the Kalman gain is found K_k, which determines how much the prediction should be corrected based on the new measurement:

$$\boldsymbol{K}_k = \boldsymbol{P}_{k+1}^{-} \boldsymbol{H}_k^T (\boldsymbol{H}_k \boldsymbol{P}_{k+1}^{-} \boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1} \ldots\ldots\ldots (5.2)$$

Where:

- $\boldsymbol{H}_k$ is the observation matrix of (3×1).
- $\boldsymbol{K}_k$ is the Kalman gain matrix of (3×3), a blending factor for updating the estimate using measurements.
- $\boldsymbol{P}_{k+1}^{-}$ is the priori predicted error covariance matrix of (3×3).
- $\boldsymbol{R}_k$is the covariance matrix of (1×1) of the measurement noise $\boldsymbol{v}_k$.

3. Update step

In this step many variables are updated subsequently such as $\widehat{\boldsymbol{x}}_{k+1}$, $\boldsymbol{P}_{k+1}$, $\boldsymbol{e}_k$ also the filter incorporates the new measurement z_k to correct the prediction.

The state estimate is then updated:

$$\widehat{\boldsymbol{x}}_{k+1} = \widehat{\boldsymbol{x}}_{k+1}^{-} + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{H}_k\widehat{\boldsymbol{x}}_{k+1}^{-}) \dots\dots\dots (6.2)$$

Where:

- $\widehat{\boldsymbol{x}}_{k+1}$ is the posterior predicted state matrix of (3×1).
- $\widehat{\boldsymbol{x}}_{k+1}^{-}$ is the priori predicted state matrix of (3×1).
- $\boldsymbol{K}_k$ is the Kalman gain matrix of (3×3), a blending factor for updating the estimate using measurements.
- $\boldsymbol{z}_k$ is the observation (measurement) vector of (1×1)at time t_k.
- $\boldsymbol{H}_k$ is the observation matrix of (1×3), mapping the true state to the observed state.

The error covariance matrix is updated to reflect the improved estimate:

$$\boldsymbol{P}_{k+1} = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{P}_{k+1}^{-} \dots\dots\dots (7.2)$$

Where:

- $\boldsymbol{P}_{k+1}$ is the predicted error covariance matrix of (3 × 3).
- $\boldsymbol{K}_k$ is the Kalman gain matrix of (3 × 3), a blending factor for updating the estimate using measurements.
- $\boldsymbol{H}_k$ is the observation matrix of (1 × 3), mapping the true state to the observed state.
- $\boldsymbol{P}_{k+1}^{-}$ is the priori predicted error covariance matrix of (3 × 3).

Also we have to find the error matrix to actually find how good the filter is working, so it can be found out by

$$\boldsymbol{e}_k = \boldsymbol{x}_k - \widehat{\boldsymbol{x}}_k \ldots\ldots\ldots (8.2)$$

Where:

- $\boldsymbol{e}_k$ is the error matrix of (3 × 1).
- $\boldsymbol{x}_k$ is the actual value of the state vector of (3 × 1).
- $\widehat{\boldsymbol{x}}_k$ is the estimated value of the state vector of (3 × 1).

4. Kalman filter algorithm

The algorithm works exactly in the same way as already been shown in sub-part (4) of section (3.2).

5. Flowchart

The flowchart is also the same as shown in Fig. (1).

3.5 N-dimensional Kalman filter

An N-dimensional Kalman filter is a generalized version of the Kalman filter used to estimate the state of a system with N variables or dimensions, where N can be any number based on the complexity of the system. Like other Kalman filters, it works by blending predictions from a model with noisy measurements to produce more accurate estimates over time. Let us understand with a simple example, consider a complex weather monitoring system where N variables such as temperature, humidity, wind speed, and atmospheric pressure are being tracked at various locations. Each of these variables can be noisy due to sensor inaccuracies. The N-dimensional Kalman filter helps by:

- Predicting the future values of these N weather variables based on a weather model.
- Updating the predicted values with real-time sensor data, balancing the prediction with the noisy measurements to refine the estimates.

With each new set of data, the filter continuously improves its estimate of the N variables, offering more accurate and reliable monitoring of the weather conditions. It can be used in complex systems, such as in robotic systems, sensor networks etc. The working principle is exactly the same as the previous two-dimensional or three-dimensional Kalman filters. The major changes are in the dimension of all the matrices.

In an N-dimensional system, the state vector is:

$$\boldsymbol{x}_k = \begin{bmatrix} x_{1,k} \\ x_{2,k} \\ . \\ x_{N,k} \end{bmatrix} \dots\dots\dots (11)$$

Where $x_{1,k}$ is the first state, $x_{2,k}$ is the second state and $x_{N,k}$ is the Nth state.

Again, the filter follows two key steps prediction and update. Before that we again need to come up with the discrete time domain input and output equations of the given plant. Here it is represented as follows -

$$\boldsymbol{x}_{k+1} = \boldsymbol{F}_k \boldsymbol{x}_k + \boldsymbol{w}_k \dots\dots\dots (1.3)$$

$$\boldsymbol{y}_k = \boldsymbol{H}_k \boldsymbol{x}_k + \boldsymbol{v}_k \dots\dots\dots (2.3)$$

Where:

- $\boldsymbol{x}_{k+1}$ is the predicted state matrix of $(n \times 1)$.
- $\boldsymbol{F}_k$ is the state transition matrix of $(n \times n)$, describing how the state evolves from time step k to $k+1$.
- $\boldsymbol{x}_k$ is the state vector of $(n \times 1)$ at time t_k.
- $\boldsymbol{w}_k$ is the process noise (Gaussian white noise) vector of $(n \times 1)$.
- $\boldsymbol{y}_k$ is the output state of (1×1).
- $\boldsymbol{H}_k$ is the observation matrix of $(1 \times n)$.
- $\boldsymbol{v}_k$ is the measurement noise (Gaussian white noise) matrix of (1×1).

1. Prediction step

The filter predicts the next state of the system using a model of the system's dynamics. It also projects the error covariance matrix, which quantifies the uncertainty in the prediction. The state prediction can be described by:

$$\widehat{\boldsymbol{x}}_{k+1}^{-} = \boldsymbol{F}_k \widehat{\boldsymbol{x}}_k \ldots\ldots\ldots (3.3)$$

Where:

- $\widehat{\boldsymbol{x}}_{k+1}^{-}$ is the predicted state vector of $(n \times 1)$.
- $\boldsymbol{F}_k$ is the state transition matrix of $(n \times n)$, describing how the state evolves from time step k to $k + 1$.
- $\boldsymbol{x}_k$ is the state vector of $(n \times 1)$ at time t_k.

The error covariance matrix, $\boldsymbol{P}_k$, is updated as:

$$\boldsymbol{P}_{k+1}^{-} = \boldsymbol{F}_k \boldsymbol{P}_k \boldsymbol{F}_k^T + \boldsymbol{Q}_k \ldots\ldots\ldots (4.3)$$

Where:

- $\boldsymbol{P}_{k+1}^{-}$ is the priori predicted error covariance matrix of $(n \times n)$.
- $\boldsymbol{F}_k$ is the state transition matrix of $(n \times n)$, describing how the state evolves from time step k to $k + 1$.
- $\boldsymbol{P}_k$ is the error covariance matrix of $(n \times n)$ associated with the estimate $\widehat{\boldsymbol{x}}_k$.
- $\boldsymbol{Q}_k$ is the covariance matrix of $(n \times n)$ the process noise $\boldsymbol{w}_k$.
- $\boldsymbol{w}_k$ is the process noise (Gaussian white noise) vector of $(n \times 1)$.

2. Determination of Kalman gain

In this step, the Kalman gain is found K_k, which determines how much the prediction should be corrected based on the new measurement:

$$\boldsymbol{K}_k = \boldsymbol{P}_{k+1}^{-}\boldsymbol{H}_k^T(\boldsymbol{H}_k\boldsymbol{P}_{k+1}^{-}\boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1} \dots\dots\dots (5.3)$$

Where:

- $\boldsymbol{H}_k$ is the observation matrix of $(n \times 1)$.
- $\boldsymbol{K}_k$ is the Kalman gain matrix of $(n \times n)$, a blending factor for updating the estimate using measurements.
- $\boldsymbol{P}_{k+1}^{-}$ is the priori predicted error covariance matrix of $(n \times n)$.
- $\boldsymbol{R}_k$is the covariance matrix of (1×1) of the measurement noise $\boldsymbol{v}_k$.

3. Update step

In this step many variables are updated subsequently such as $\widehat{\boldsymbol{x}}_{k+1}$, $\boldsymbol{P}_{k+1}$, $\boldsymbol{e}_k$ also the filter incorporates the new measurement z_k to correct the prediction.

The state estimate is then updated:

$$\widehat{\boldsymbol{x}}_{k+1} = \widehat{\boldsymbol{x}}_{k+1}^{-} + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{H}_k\widehat{\boldsymbol{x}}_{k+1}^{-}) \dots\dots\dots (6.3)$$

Where:

- $\widehat{\boldsymbol{x}}_{k+1}$ is the posterior predicted state matrix of $(n \times 1)$.
- $\widehat{\boldsymbol{x}}_{k+1}^{-}$ is the priori predicted state matrix of $(n \times 1)$.
- $\boldsymbol{K}_k$ is the Kalman gain matrix of $(n \times n)$, a blending factor for updating the estimate using measurements.
- $\boldsymbol{z}_k$ is the observation (measurement) vector of (1×1)at time t_k.

- $\boldsymbol{H}_k$ is the observation matrix of $(1 \times n)$, mapping the true state to the observed state.

The error covariance matrix is updated to reflect the improved estimate:

$$\boldsymbol{P}_{k+1} = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k)\boldsymbol{P}_{k+1}^{-} \dots\dots\dots (7.3)$$

Where:

- $\boldsymbol{P}_{k+1}$ is the predicted error covariance matrix of $(n \times n)$.
- $\boldsymbol{K}_k$is the Kalman gain matrix of $(n \times n)$, a blending factor for updating the estimate using measurements.
- $\boldsymbol{H}_k$ is the observation matrix of $(1 \times n)$, mapping the true state to the observed state.
- $\boldsymbol{P}_{k+1}^{-}$ is the priori predicted error covariance matrix of $(n \times n)$.

Also we have to find the error matrix to actually find how good the filter is working, so it can be found out by

$$\boldsymbol{e}_k = \boldsymbol{x}_k - \widehat{\boldsymbol{x}}_k \dots\dots\dots (8.3)$$

Where:

- $\boldsymbol{e}_k$ is the error matrix of $(n \times 1)$.
- $\boldsymbol{x}_k$ is the actual value of the state vector of $(n \times 1)$.
- $\widehat{\boldsymbol{x}}_k$ is the estimated value of the state vector of $(n \times 1)$.

4. Kalman filter algorithm

The algorithm works exactly in the same way as already been shown in sub-part (4) of section (3.2).

5. Flowchart

The flowchart is also the same as shown in Fig. (1).

3.6 Case study : Application of KF for a 3D linear system

The Kalman filter is specifically designed for linear systems and works effectively when the system's dynamics and the measurement process can be described using linear models. A linear system is one where the relationship between variables is proportional, meaning that the system's behaviour can be described using linear equations. Here the variables are not interdependent on other variables nor are function of any other variable. For linear systems, the Kalman filter operates efficiently by utilizing the linear relationship between the system's current state, its previous state, and the measurements. Here's how it functions in a linear system:

- **System model:**

We have taken a linear system with all its parameters as follows:

$$F = \begin{bmatrix} 0.9 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 0.7 \end{bmatrix}$$

$$H = [1 \quad 0.1 \quad 0.2]$$

$$Q = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}$$

$$R = [1]$$

$$P_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$x_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

- **Filter parameters:**

$$F_f = \begin{bmatrix} 0.9 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 0.7 \end{bmatrix}$$

$$H_f = [1 \quad 0.1 \quad 0.2]$$

$$Q_f = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}$$

$$R_f = [1]$$

The Kalman filter assumes that the current state of the system depends linearly on the previous state and possibly some control inputs. We have taken the above and has developed a MATLAB code. This code has been so developed that it can be used for one-dimensional, two-dimensional, three-dimensional and N-dimensional systems. Just by changing the order number and according to the order number the required input matrices has to be changed accordingly then we are good to go. Here are some responses when we had got, the system parameters and filter parameters are known, moreover we are making the filter parameters same as that of the filter parameters so that we can see whether the Kalman filter is actually working by tracing the original response of the system.

Here are some of the results for the above system:

1. Here is a result of a known 3rd order system with a time up to 60 seconds and 5000 Monte Carlo runs.

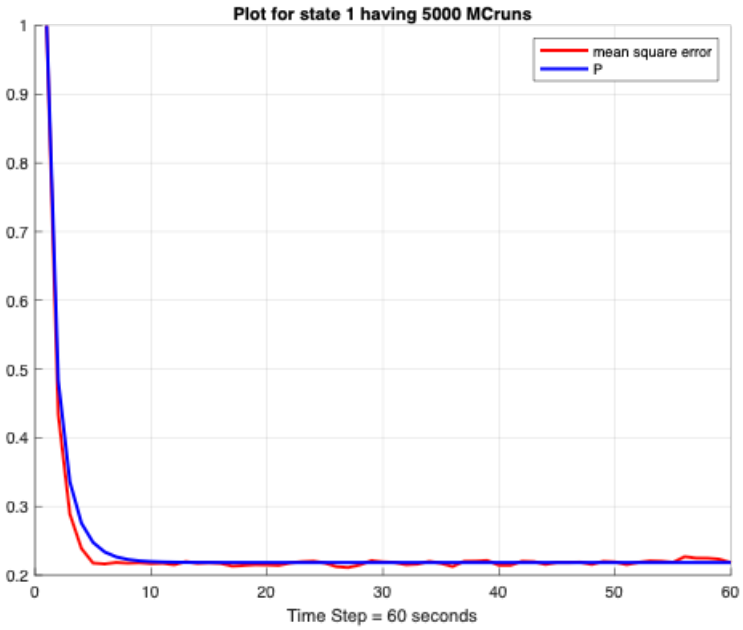

Fig.2

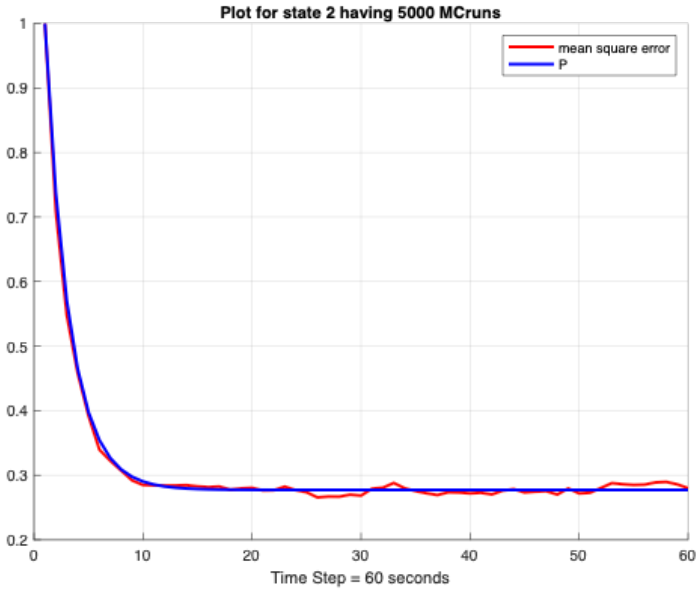

Fig.3

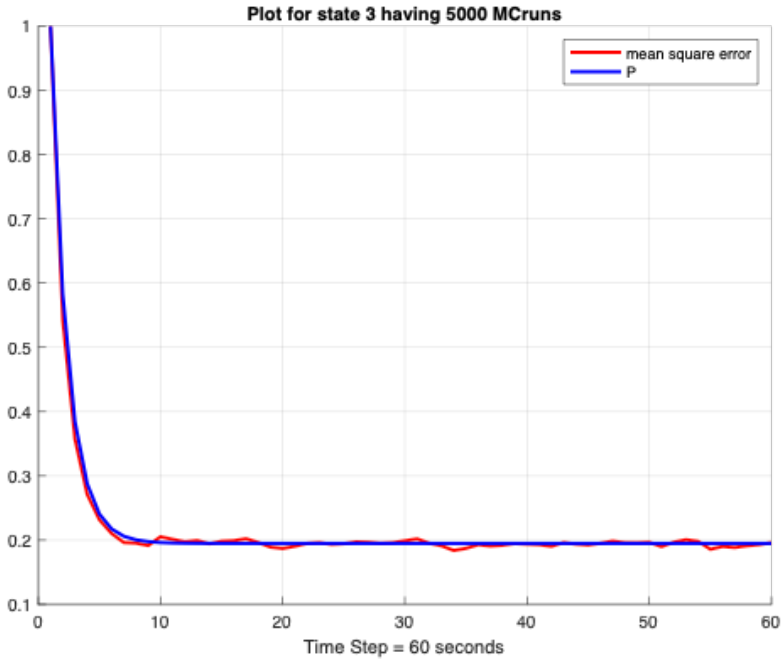

Fig.4

2. Here is the result of the same known 3rd order system with a time up to 60 seconds and 15000 Monte Carlo runs.

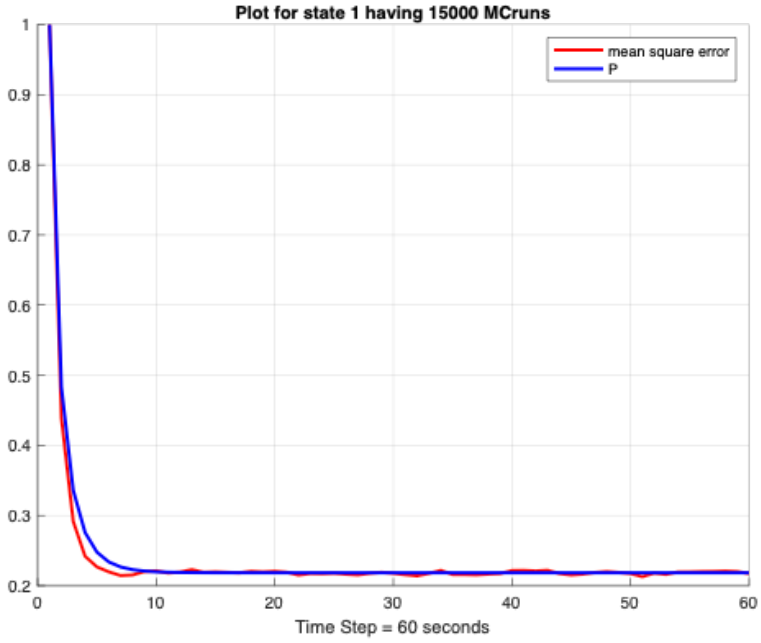

Fig.5

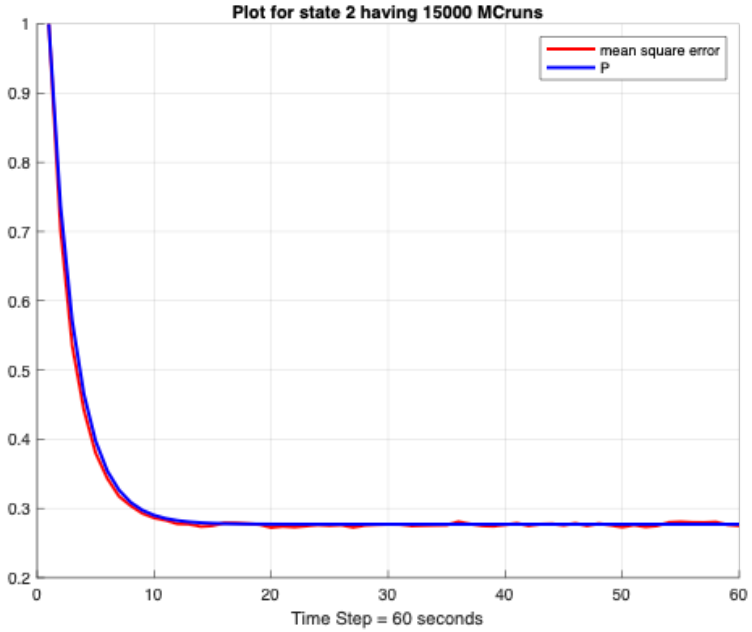
Plot for state 2 having 15000 MCruns
mean square error
P
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
50
60
Time Step = 60 seconds

Fig.6

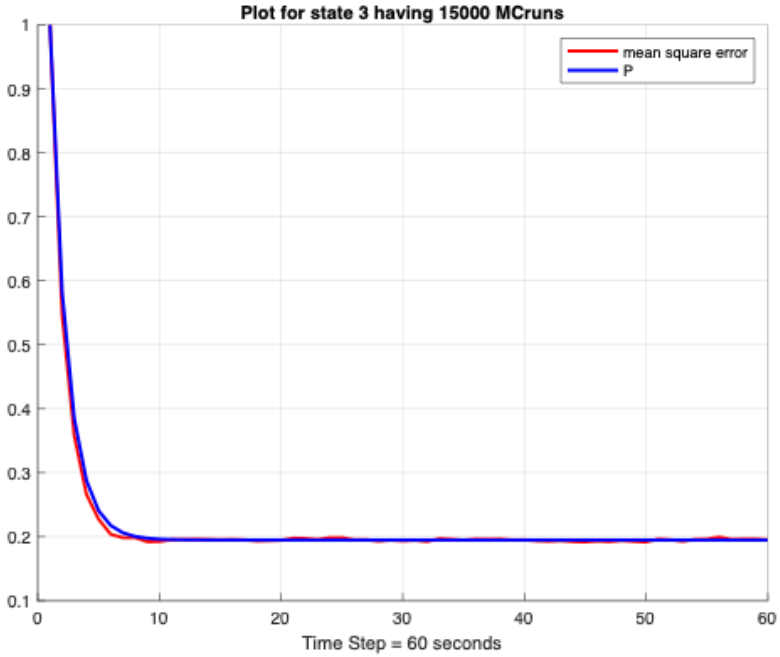
Plot for state 3 having 15000 MCruns
mean square error
P
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50
60
Time Step = 60 seconds

Fig.7

From the above results, it is clear that the Kalman filter performs effectively with a known linear system. Moreover, we can adjust or fine-tune the filter parameters based on our requirements and observe how these changes influence its behaviour. It is also noteworthy that increasing the number of Monte Carlo runs from 5,000 to 15,000 smoothens the ripples, and the predicted state closely aligns with the actual state.

In a relatively short time, the Kalman filter successfully traces the system's actual state, even with the presence of both measurement and process noise. In the next chapter, we will extend this analysis to nonlinear systems using the Extended Kalman Filter (EKF) and the recently proposed Adaptive Kalman Filter (AKF) to evaluate their performance and potential improvements.

Chapter 4 - Brief overview of Extended Kalman filter (EKF)

4.0 Introduction

The Extended Kalman Filter (EKF) is an advanced version of the Kalman Filter, designed to handle nonlinear systems. While the standard Kalman Filter is optimal for systems with linear dynamics, the EKF extends its functionality to nonlinear processes, making it particularly useful in many real-world applications, such as robotics, navigation, and battery state-of-charge (SOC) estimation. In nonlinear systems, both the state dynamics and measurement processes may involve nonlinear relationships. The Extended Kalman Filter (EKF) tackles this by approximating nonlinear functions through linearization. At each step, it employs a first-order Taylor series expansion around the current estimate to provide an approximation of the nonlinear functions. This allows the EKF to perform the standard Kalman Filter's prediction and update steps but applied to a locally linearized version of the system. The Extended Kalman Filter (EKF) is a widely used technique for state estimation in nonlinear systems, and in the paper [Proctor2020], it is applied to estimate the State-of-Charge (SOC) of rechargeable batteries using the Nonlinear Double-Capacitor (NDC) model. The NDC model is a novel equivalent circuit model (ECM) that simulates charge diffusion and nonlinear voltage behaviour in batteries, offering better accuracy in predicting SOC. The Extended Kalman Filter (EKF) is mostly used because of its computational efficiency and ability to track the state of nonlinear systems in real time. However, it relies on the accuracy of the

linearization process, which may not be sufficient for highly nonlinear systems, but it works effectively when the nonlinearity is mild, as in many engineering applications.

The EKF can be represented as follows –

$$x_{k+1} = f(x_k, u_k) + w_k \dots\dots\dots (12)$$

$$y_k = h(x_k, u_k) + v_k \dots\dots\dots (13)$$

Where:

- x_k is the state vector at time k.
- x_{k+1} is the predicted state.
- $f(x_k, u_k)$ is the nonlinear function of state & input at time k.
- u_k is the input.
- y_k is the measured output.
- w_k is the process noise which is assumed to be Gaussian with covariance Q.
- v_k is the measurement noise, assumed to be Gaussian with covariance R.
- $h(x_k, u_k)$ is the nonlinear function of state & input at time k.
- For the state transition function $f(x)$, the Jacobian is:

$$F_k = \frac{\partial f}{\partial x} \mid_{\hat{x}_{k-1|k-1}}$$

- For the measurement function $h(x)$, the Jacobian is:

$$H_k = \frac{\partial h}{\partial x} \mid_{\hat{x}_{k|k-1}}$$

4.1 Application of EKF for SOC estimation

Here we have taken the Nonlinear Double-Capacitor (NDC) model that simulates charge diffusion and nonlinear voltage behaviour in batteries, offering better accuracy in predicting SOC. Using discrete time domain state space form the system dynamics are represented. However, the model we are working with has linear process equation and nonlinearity only in the output equation [Proctor2020].

$$x_{k+1} = Fx_k + Gu_k + w_k \dots\dots\dots (14)$$

$$y_k = h(x_k, u_k) + v_k \dots\dots\dots (15)$$

given that –

$$F = e^{AT} \dots\dots\dots (16)$$

$$G = (\int_0^T e^{AT}\, d\tau) * B \dots\dots\dots (17)$$

Where:

- x_k is the state vector at time k.
- u_k is the input current.
- y_k is the measured voltage.
- w_k is the process noise which is assumed to be Gaussian with covariance Q.
- v_k is the measurement noise, assumed to be Gaussian with covariance R.
- $h(x_k, u_k)$ is the nonlinear function of state & input at time k.

4.1.1 Problem statement: Nonlinear double capacitor model

The non-linear double capacitor model we have considered is basically used to track three state variables $V_b(t)$, $V_s(t)$ and $V_1(t)$ at every instant of time from where we can directly infer the SOC [Proctor2020].

The NDC model and its state space representation is given by–

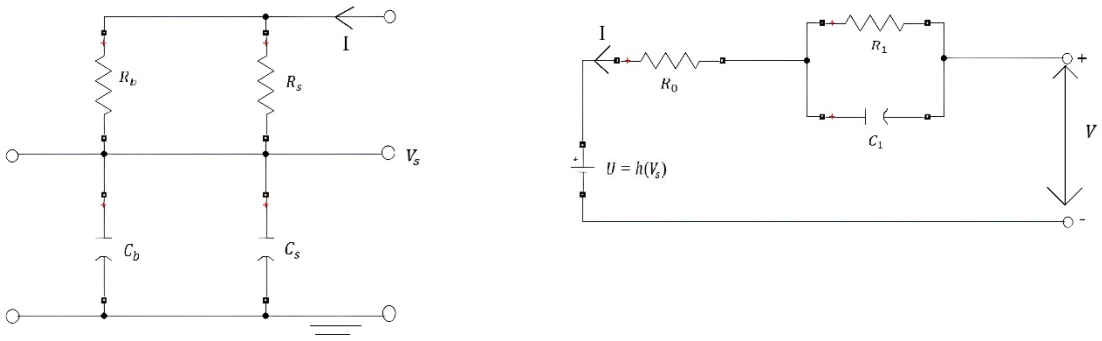

Fig. 8 Non-linear double capacitor model [Proctor2020]

❖ **Input equation**:

$$\begin{bmatrix} \dot{V}_b(t) \\ \dot{V}_s(t) \\ \dot{V}_1(t) \end{bmatrix} = A \begin{bmatrix} V_b(t) \\ V_s(t) \\ V_1(t) \end{bmatrix} + BI(t) \dots\dots\dots (18)$$

❖ **Output equation**:

$$V(t) = h\big(V_s(t)\big) - V_1(t) + R_0 I(t) \dots\dots\dots (19)$$

❖ **State variables**:

- $V_b(t)$: Voltage across the bulk region of the electrode in the nonlinear double-capacitor (NDC) model.

- $V_s(t)$: Voltage across the surface region of the electrode in the NDC model.
- $V_1(t)$: Voltage across the resistor-capacitor (RC) circuit in the NDC model.

❖ **Input variable**:

- $I(t)$: Input current at time t, representing the current flowing through the battery.

❖ **System matrices**:

- A: The system matrix, which defines the relationships between the state variables $V_b(t)$, $V_s(t)$, and $V_1(t)$ in terms of their rates of change.
- B: The input matrix which defines how the input current $I(t)$ influences the state variables.

❖ **Nonlinear function**:

- $h\big(V_s(t)\big)$: A nonlinear function representing the open-circuit voltage (OCV) as a function of the surface voltage $V_s(t)$. It accounts for the nonlinear relationship between the SOC and the terminal voltage of the battery.

❖ **Output variable**:

- $V(t)$: The measured terminal voltage of the battery, which is a combination of the nonlinear OCV function $h\big(V_s(t)\big)$, the internal voltage $V_1(t)$, and the product of the internal resistance R_0 and the input current $I(t)$.

❖ **Resistance**:

- R_0: The internal resistance of the battery, which affects the voltage drop due to the input current $I(t)$.

These equations together form the core of the battery's dynamic model in the context of state-of-charge (SOC) estimation using the Extended Kalman filter (EKF). The matrices A and B used in equation (18) in the state-space model of the Nonlinear Double-Capacitor (NDC) model are as follows [Proctor2020]:

Matrix A:

$$A = \begin{bmatrix} -\frac{1}{C_b(R_b+R_s)} & \frac{1}{C_b(R_b+R_s)} & 0 \\ \frac{1}{C_s(R_b+R_s)} & -\frac{1}{C_s(R_b+R_s)} & 0 \\ 0 & 0 & -\frac{1}{R_1C_1} \end{bmatrix} \dots\dots\dots (20)$$

- C_b: Capacitance representing the bulk portion of the battery electrode.
- C_s: Capacitance representing the surface region of the electrode.
- R_b: Resistance in the bulk region of the battery electrode.
- R_s: Resistance in the surface region of the battery electrode.
- R_1: Internal resistance in the RC branch.
- C_1: Capacitance in the RC branch.

Matrix B:

$$B = \begin{bmatrix} \frac{R_s}{C_b(R_b + R_s)} \\ \frac{R_b}{C_s(R_b + R_s)} \\ -\frac{1}{C_1} \end{bmatrix} \dots\dots\dots (21)$$

- The elements in matrix B show how the input current $I(t)$ influences the voltages $V_b(t)$, $V_s(t)$, and $V_1(t)$.

❖ **Matrix** A governs the internal dynamics of the system, representing how the voltages across the different capacitors change over time.

❖ **Matrix** B represents how the input current $I(t)$ affects the state variables (voltages) in the system.

These matrices are crucial components in the state-space representation of the NDC model, used to estimate the (SOC) via the extended Kalman filter (EKF). The above system we are working with is non-linear in nature which can be determined from the measurement output equation (19). Here we are going to discuss about how to calculate the SOC and the non-linearity and how we can linearize it. In the non-linear measurement equation, the $h(V_s)$ is the non-linear output function determining the nonlinear behaviour of the voltage which is parameterized through a second-order polynomial [Proctor2020].

$$SOC = \frac{C_b V_b + C_s V_s}{C_b + C_s} \times 100\% \dots\dots\dots (22)$$

$$h(V_s) = \alpha_0 + \alpha_1 V_s + \alpha_2 V_s^2 \dots\dots\dots (23)$$

Here, the value of V_s has been set in between $0 < V_s \leq 1$.

4.1.2 EKF formulation

The model's low dimensionality and nonlinearity makes the Extended Kalman Filter (EKF) a suitable choice. Since the system only has three dimensions, applying the EKF is relatively simple. The main challenge comes from the nonlinear measurement equation, and the EKF's linearization effectively addresses this issue, making it an efficient solution Since, the NDC model consists of only three states, enabling efficient computation when applying the EKF. Moreover, the model's nonlinearity is limited to the measurement process, which simplifies the linearization and makes implementing the EKF easier. This approach is then thoroughly tested through simulations and experiments. Alongside this we need to keep in mind that if the non-linearity is much more than desired then EKF might fail to incorporate accurately results, in that scenario we have to further reach out for other versions of Kalman filters available which will be able to give us better results in this respect.

The EKF algorithm is exactly same as the KF algorithm with only the introduction of the non-linear function which we will be looking into. Apart from that the rest of the algorithm is same as that of the KF. It majorly includes three steps that is to be followed –

- Prediction:

$$\hat{x}_{k+1}^{-} = F_k \hat{x}_k + G u_k \dots\dots\dots (24)$$

$$P_{k+1}^{-} = F_k P_k F_k^T + Q_k \dots\dots\dots (25)$$

- Measurement update & Kalman gain:

$$\hat{x}_{k+1} = \hat{x}_{k+1}^{-} + K_k (z_k - H_k \hat{x}_{k+1}^{-}) \dots\dots\dots (26)$$

Where the Kalman Gain K_k is computed as:

$$K_k = P_{k+1}^{-} H_k^T (H_k P_{k+1}^{-} H_k^T + R_k)^{-1} \dots\dots\dots (27)$$

Here, H_k is the Jacobian matrix of the measurement function $h(x_k)$,

which is calculated as:

$$H_k = \frac{\partial h}{\partial x} \mid_{x=\hat{x}_{k|k-1}} \dots\dots\dots (28)$$

- Update of error covariance:

$$P_{k+1} = (I - K_k H_k) P_{k+1}^{-} \dots\dots\dots (29)$$

Now the main thing to do is linearize the above nonlinear polynomial. This can be done with the help of Jacobian.

❖ **The Jacobian matrix:**

In control systems, the Jacobian refers to a matrix of partial derivatives that represents how small changes in input variables affect the output of a system. It is particularly useful when dealing with nonlinear systems, where the relationship between inputs and outputs isn't linear.

For a nonlinear system described by a vector of state equations **x** and outputs **y**, the Jacobian matrix provides a linear approximation of the system's dynamics around a specific operating point. It helps in understanding how the system behaves locally near that point.

In the context of a system defined by:

- State equations: $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$
- Output equations: $\mathbf{y} = g(\mathbf{x}, \mathbf{u})$

Where:

- **x** is the state vector
- **u** is the input vector
- f and g are nonlinear functions

❖ **Jacobian of the state equation**:

- $A = \frac{\partial f}{\partial \mathbf{x}}$, which describes how the states change with small variations in the state vector.
- $B = \frac{\partial f}{\partial \mathbf{u}}$, which describes how the states change with small variations in the input vector.

❖ **Jacobian of the output equation**:

- $C = \frac{\partial g}{\partial \mathbf{x}}$, which describes how the outputs change with small variations in the state vector.
- $D = \frac{\partial g}{\partial \mathbf{u}}$, which describes how the outputs change with small variations in the input vector.

In control theory, the Jacobian matrix is especially important in linearizing nonlinear systems around a certain operating point, which allows the use of linear control techniques like the Kalman filter. The Jacobian matrix H_k is the partial derivative of the measurement function $h(x)$ with respect to the state vector x. It linearizes the nonlinear measurement function around the current state estimate. In this case, the measurement function $h(x)$ is related to the battery voltage, which depends on the surface voltage V_s, and the internal state variables V_b, V_s, V_1 [Proctor2020].

The Jacobian matrix H_k is given by [Proctor2020]:

$$H_k = \frac{\partial h}{\partial x} \mid_{x=\hat{x}_{k|k-1}} = \left[\frac{\partial h(V_s)}{\partial V_b} \quad \frac{\partial h(V_s)}{\partial V_s} \quad \frac{\partial h(V_s)}{\partial V_1}\right]^T \ldots\ldots\ldots (30)$$

For the NDC model, the specific form of H_k is [Proctor2020]:

$$\begin{bmatrix} \left(-\gamma_2\gamma_3 e^{-\gamma_3 SOC} + \gamma_4\gamma_5 e^{-\gamma_5(1-SOC)}\right)\frac{I(k)C_b}{Q_t} \\ \frac{\partial h(V_s)}{\partial V_s} + \left(-\gamma_2\gamma_3 e^{-\gamma_3 SOC} + \gamma_4\gamma_5 e^{-\gamma_5(1-SOC)}\right)\frac{I(k)C_s}{Q_t} \\ -1 \end{bmatrix} \ldots\ldots\ldots (31)$$

Where:

- SOC is the state of charge, which is dependent on V_b and V_s.
- $h(V_s)$ is a nonlinear function of V_s, the surface voltage.
- $\frac{\partial h(V_s)}{\partial V_s} = \alpha_1 + 2\alpha_2 V_s$ is the derivative of the nonlinear function representing the open-circuit voltage (OCV).
- $\gamma_2, \gamma_3, \gamma_4, \gamma_5$ are the parameters of the internal resistance model that depends on the SOC.
- $I(k)$ is the input current at time k.
- C_b, C_s are the capacitances of the capacitors.
- Q_t is the total battery capacity.

The Jacobian matrix H_k linearizes the nonlinear measurement function, allowing the EKF to handle the nonlinearity in the system efficiently. This approach ensures accurate estimation of the battery's SOC using the NDC model and the extended Kalman filter.

Chapter 5 - Recently proposed Adaptive Kalman Filter (AKF)

5.0 Introduction

To deal with mild nonlinearity we have seen that EKF performs quiet well but in the nonlinear model we are working with EKF is failing to converge or reduce the root mean square error. This is very natural as if the nonlinearity is of strict nature EKF do fail at times for such scenarios some other variations of Kalman filters had been developed such as Sigma Point Kalman Filter, Unscented Kalman Filter, Square-root unscented Kalman Filter, Adaptive Square root unscented Kalman Filter and etc. Here we are going to implement a recently proposed Adaptive Kalman Filter [Takayama2024].

The paper [Takayama2024] introduces a new method to improve the performance of the Kalman filter, which is commonly used for estimating the state of a system, such as the position or speed of a moving object. The Kalman filter works by combining measurements from sensors with a mathematical model of the system. However, when the model or measurements are not accurate, the filter can make poor estimates or even fail, a problem known as "filter divergence."

The proposed solution focuses on adjusting the "process noise," which accounts for uncertainties in the system model. Traditional approaches to this problem add extra noise to the system manually, which can sometimes make the filter overly cautious, leading to large errors in the estimates. This is especially problematic when the number of measurements is smaller than the number of

variables being estimated, which is common in complex systems like autonomous vehicles or robots navigating through environments with changing conditions.

What makes this new method different is that it adapts the process noise dynamically, based on the structure of the measurement matrix—the matrix that links the system's state to the measurements. The measurement matrix changes depending on the quality and quantity of the available measurements, such as when a robot moves into an area with fewer sensors or a vehicle loses satellite signals.

The paper's approach involves using this measurement matrix to adjust the process noise at each time step, ensuring that the filter remains flexible and avoids becoming overly cautious or aggressive in its estimates. Unlike previous methods, this technique does not rely on the measurements themselves to make adjustments but uses the structure of the matrix to ensure that errors are minimized. This leads to more accurate state estimates without the unnecessary inflation of estimation errors that can occur in traditional methods.

We are mainly interested in the algorithm of the recently proposed Adaptive Kalman Filter which we are going to implement in the nonlinear system that we are working on and our main objective is to check whether the recently proposed Adaptive Kalman Filter gives better results as compared to that of the Extended Kalman Filter which had been suggested in the paper [Proctor2020].

This is a smarter way to handle uncertainties in system modeling and measurement variations, leading to more reliable and accurate state estimation in complex, real-world systems.

5.1 Working principle

We had already discussed earlier in Chapter 3 the working of the Kalman filter, the recently proposed adaptive Kalman filter is nothing different from it. Just the only major difference is in the algorithm that we had taken from paper [Takayama2024], which we are going to discuss here.

The main objective here is to get the results improved with respect to the Extended Kalman filter that we had discussed in Chapter 4. The adaptive Kalman filter proposed in the paper [Takayama2024] improves the performance of the traditional Extended Kalman Filter (EKF) by dynamically adjusting the process noise covariance based on the measurement matrix. The adaptive Kalman filter introduced in the paper [Takayama2024] tackles these issues by adjusting the process noise covariance dynamically at each time step. The novelty here is that this adjustment depends on the measurement matrix H_k, which describes how the system's state relates to the measurements, rather than the measurements themselves. The main idea is to adapt the process noise in such a way that it reflects the current measurement configuration, improving filter performance in changing environments.

Now let us understand how the recently proposed Adaptive Kalman filter works [Takayama2024].

- **Measurement matrix** H_k: The filter continuously monitors the measurement matrix, which changes depending on the quality and quantity of the measurements at each time step.
- **Fictitious Noise** δQ_k: At each time step, the filter adds an optimal amount of *fictitious noise* to the process noise covariance. This fictitious noise

helps account for uncertainties in the model that are not captured by the fixed process noise.

- **Dynamic Adjustment**: The fictitious noise δQ_k is computed based on the measurement matrix H_k. Specifically, the fictitious noise is chosen to minimize the expected value of the measurement residuals (the difference between actual and predicted measurements). By doing this, the filter ensures that it does not unnecessarily inflate the estimation error covariance, which would make the estimates less accurate.
- **Kalman Gain Update**: The Kalman gain, which controls how much the state estimate should be adjusted based on the new measurements, is also updated to incorporate the fictitious noise. This ensures that the filter remains responsive to changes while avoiding over-correction when measurements are sparse or unreliable.
- **State Update**: With the updated Kalman gain and the innovation (difference between the actual and predicted measurements), the filter adjusts the state estimate. This step is similar to the traditional Kalman filter, but now the process noise has been dynamically adapted to reflect the current measurement conditions.

5.2 AKF with nonlinear systems

Here, we are taking the same Nonlinear Double-Capacitor (NDC) model that simulates nonlinear voltage behaviour in batteries, offering better accuracy in predicting SOC which we had already discussed in Chapter 4. Now we are going to discuss how the recently proposed adaptive Kalman filter algorithm has been implemented using MATLAB.

Note: Since it is a third order model so the matrices will be defined accordingly.

Update of Covariance [Takayama2024]**:**

$$P_k^-(\delta Q_k^*) = F_{k-1}P_{k-1}^+F_{k-1}^T + Q_k + \delta Q_k^* \dots\dots\dots (32)$$

- $P_k^-(\delta Q_k^*)$ is the prior estimation error covariance matrix after adding optimal fictitious noise.
- $F_{k-1}$is the Jacobian of the system model.
- P_{k-1}^+ is the posterior estimation error covariance matrix.
- Q_k is the process noise covariance matrix.
- δQ_k^* is the optimal fictitious noise covariance matrix.

Kalman Gain [Takayama2024]**:**

$$K_k = P_{k+1}^- H_k^T (H_k P_{k+1}^- H_k^T + R_k)^{-1} \dots\dots\dots (33)$$

Where:

- H_k is the observation matrix.
- K_k is the Kalman gain matrix.
- P_{k+1}^- is the priori predicted error covariance matrix.
- R_k is the covariance matrix of the measurement noise $\boldsymbol{v}_k$.

Prediction Step [Takayama2024]**:**

$$\widehat{\boldsymbol{x}}_{k+1}^- = \boldsymbol{F}_k \widehat{\boldsymbol{x}}_k \dots\dots\dots (34)$$

Where:

- $\widehat{\boldsymbol{x}}_{k+1}^-$ is the predicted state vector.
- $\boldsymbol{F}_k$ is the state transition matrix, describing how the state evolves.

- $\boldsymbol{x}_k$ is the state vector.

Also,

$$O_k = H_k^T R_k^{-1} H_k \dots\dots\dots (35)$$

Where:

- H_k is the measurement matrix.
- R_k is the measurement noise covariance matrix.

Optimal Scaling Factor [Takayama2024]**:**

$$\alpha_k^* = \frac{\mathrm{tr}(O_k^2)}{\mathrm{tr}(O_k^5 P_k^- + O_k^4)} \dots\dots\dots (36)$$

Where:

- $\mathrm{tr}(A)$ is the trace of matrix A, which is the sum of its diagonal elements.
- O_k^2, O_k^4, O_k^5 are the powers of the measurement matrix O_k.
- P_k^- is the prior estimation error covariance matrix.

Optimal adjustment of the error covariance matrix [Takayama2024]**:**

$$\Delta P_k^* = \alpha_k^* O_k \dots\dots\dots (37)$$

Where:

- α_k^* is the optimal scaling factor for the measurement matrix O_k.
- $O_k = H_k^T R_k^{-1} H_k$ is the product of the transpose of the measurement matrix, the inverse of the measurement noise covariance, and the measurement matrix.

Optimal adjustment of the Kalman Gain [Takayama2024]**:**

$$\Delta K_k^* = \Delta P_k^* H_k^T R_k^{-1} \dots\dots\dots (38)$$

Where:

- H_k is the measurement matrix.
- R_k is the measurement noise covariance.
- ΔP_k^* is the updated error covariance matrix.

Computation of Fictious Noise [Takayama2024]**:**

$$\delta Q_k^* = [I - (K_k(O) + \Delta K_k^*)H_k]^{-1} \times [\Delta P_k^* - \Delta K_k^*(H_k P_k^-(O)H_k^T + R_k)\Delta K_k^T] \times [I - (K_k(O) + \Delta K_k^*)H_k]^{-T} \dots\dots\dots (39)$$

Where:

- I is the identity matrix.
- R_k is the measurement noise covariance matrix.
- ΔP_k^* is the updated error covariance matrix.
- ΔK_k^* is the adjusted Kalman gain.
- H_k is the measurement matrix.
- $K_k(O)$ is the initial Kalman gain.
- $P_k^-(O)$ is the initial covariance matrix.

Updated Kalman gain [Takayama2024]**:**

$$K_k(\delta Q_k^*) = P_k^-(\delta Q_k^*)H_k^T(H_k P_k^-(\delta Q_k^*)H_k^T + R_k)^{-1} = K_k + \Delta K_k^* \dots\dots\dots (40)$$

- $K_k(\delta Q_k^*)$ is the Kalman gain matrix with the optimal fictitious noise.
- $P_k^-(\delta Q_k^*)$ is the prior estimation error covariance matrix.
- H_k is the measurement matrix.
- R_k is the measurement noise covariance matrix.
- ΔK_k^* is the adjusted Kalman gain matrix.

- K_k is the Kalman gain matrix.

Updated posterior state estimate [Takayama2024]**:**

$$\hat{x}_k^+(\delta Q_k^*) = \hat{x}_k^- + K_k(\delta Q_k^*)[y_k - h(\hat{x}_k^-)] \dots\dots\dots (41)$$

- $\hat{x}_k^-$ is the prior state estimate vector.
- $K_k(\delta Q_k^*)$ is the Kalman gain matrix with optimal fictitious noise.
- y_k is the measurement matrix.
- $h(\hat{x}_k^-)$ is the predicted (observation) measurement matrix.

Updated error covariance matrix [Takayama2024]**:**

$$P_k^+(\delta Q_k^*) = [I - K_k(\delta Q_k^*)H_k]P_k^-(\delta Q_k^*) \dots\dots\dots (42)$$

- $P_k^+(\delta Q_k^*)$ is the posterior estimation error covariance matrix.
- I is the identity matrix.
- $K_k(\delta Q_k^*)$ is the Kalman gain matrix.
- H_k is the measurement matrix.
- $P_k^-(\delta Q_k^*)$ is the prior estimation error covariance matrix.

Updated error matrix:

$$e_k = x_k - \hat{x}_k \dots\dots\dots (43)$$

Where:

- e_k is the error matrix.
- x_k is the state vector.
- $\hat{x}_k$ is the predicted state vector.

The flowchart is as follows:

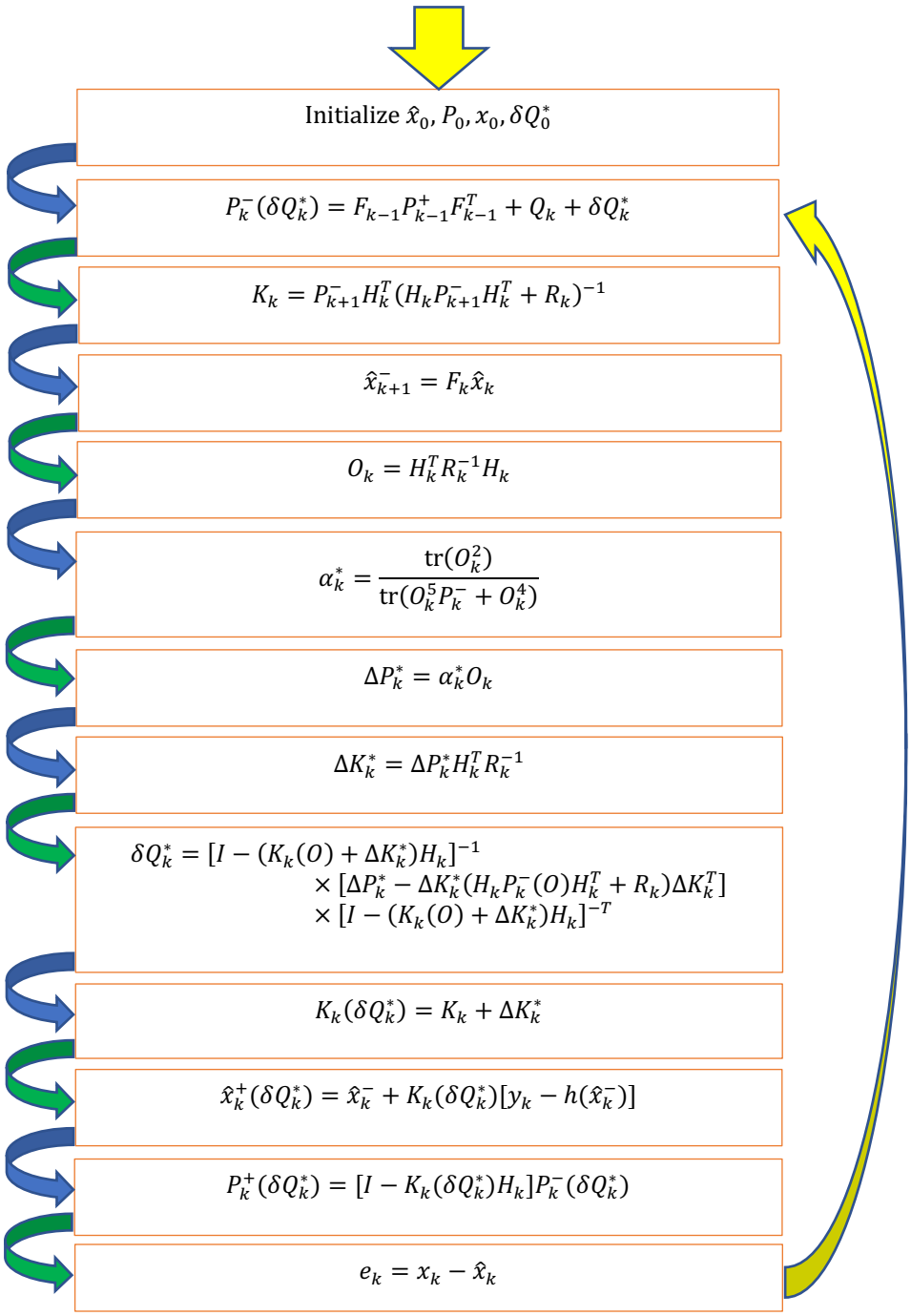

Fig. 9 The flowchart of AKF.

Chapter 6 - Comparison between EKF & AKF

6.0 Introduction

The Extended Kalman Filter (EKF) and Adaptive Kalman Filter (AKF) are critical tools for estimating states in dynamic systems, especially in scenarios where traditional linear models fall short. The EKF extends the standard Kalman Filter to handle mild nonlinearity by approximating nonlinear functions around a specific point, making it useful in applications such as robotics, navigation, and battery management. For example, it has been successfully applied in estimating the State-of-Charge (SOC) of batteries. However, the EKF's reliance on linearization makes it less accurate for systems with strong nonlinearity, where it can struggle to provide stable and precise estimates. The AKF builds upon the EKF by introducing adaptability, which allows it to dynamically adjust its noise model based on the quality and availability of measurements. Unlike the EKF, which operates with a fixed process noise, the AKF incorporates a mechanism to tune this noise according to current measurement configurations, allowing it to respond flexibly to measurement uncertainties. This adaptive nature enables the AKF to handle more complex, nonlinear systems with greater reliability, minimizing the errors that typically challenge other filters. In this chapter we are going to study a comparison between the EKF & AKF which we had already studied in the previous chapter 4 & 5 respectively.

Both EKF & AKF has been implemented on a specific system which has been taken from [Proctor2020].

The system parameters are as follows:

$C_b = 10037\ F$
$C_s = 973\ F$
$R_b = 0.019\ \Omega$
$R_s = 0\ \Omega$
$R_1 = 0.02\ \Omega$
$C_1 = 3250\ F$

Now from the above values we have calculated the following system matrices:

$$A = \begin{bmatrix} -0.0052 & 0.0052 & 0 \\ 0.0541 & -0.0541 & 0 \\ 0 & 0 & -0.0154 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0.0010 \\ -0.0003 \end{bmatrix}$$

$$G = \begin{bmatrix} 0.0000 \\ 0.0010 \\ -0.0003 \end{bmatrix}$$

$$F = \begin{bmatrix} 0.9949 & 0.0051 & 0 \\ 0.0525 & 0.9475 & 0 \\ 0 & 0 & 0.9847 \end{bmatrix}$$

$$Q = \begin{bmatrix} 0.1000 & 0 & 0 \\ 0 & 0.1000 & 0 \\ 0 & 0 & 0.1000 \end{bmatrix}$$

$$R = [0.1000]$$

Now the filter parameters are as follows:

$$F_f = \begin{bmatrix} 0.9949 & 0.0051 & 0 \\ 0.0525 & 0.9475 & 0 \\ 0 & 0 & 0.9847 \end{bmatrix}$$

$$Q_f = \begin{bmatrix} 0.1000 & 0 & 0 \\ 0 & 0.1000 & 0 \\ 0 & 0 & 0.1000 \end{bmatrix}$$

$$R_f = [0.1000]$$

6.1 Known Noise Covariance

When Noise Covariance is known, well understood and correctly specified, the filter can accurately weigh the reliability of the data, resulting in better estimates. The Kalman gain the factor determining how much the filter adjusts based on new measurements can be optimally computed. This leads to smooth and accurate tracking of the system's state.

With accurate noise covariance values, the filter can reliably produce estimates without overreacting to random fluctuations or noise. This stability is particularly beneficial for applications where consistency is more important than rapid adjustments.

When noise characteristics are known, the filter's performance is predictable, making it easier to evaluate its behaviour across different conditions. This predictability simplifies tuning the filter to achieve desired results, especially in controlled environments.

Now we will study some cases where we will see how the known noise covariance is affecting with the help of some plots-

For 300 Monte Carlo runs:

- $R = [0.1]$ & $R_f = [0.1]$

STATE 1:

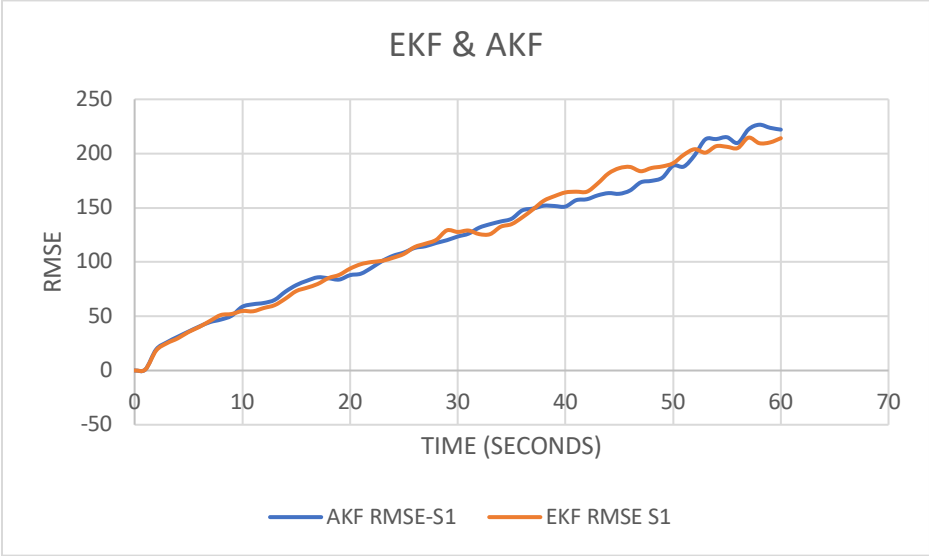

Fig.10

STATE 2:

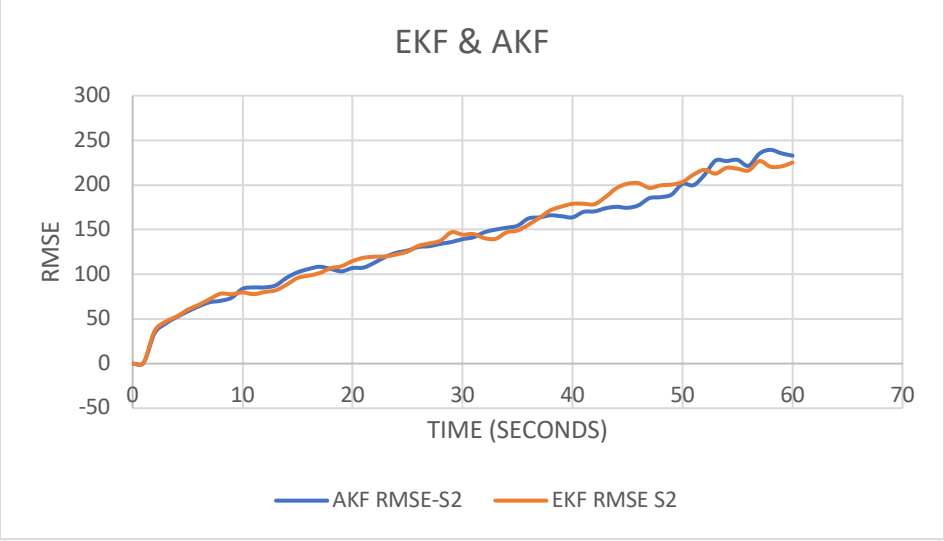

Fig.11

STATE 3:

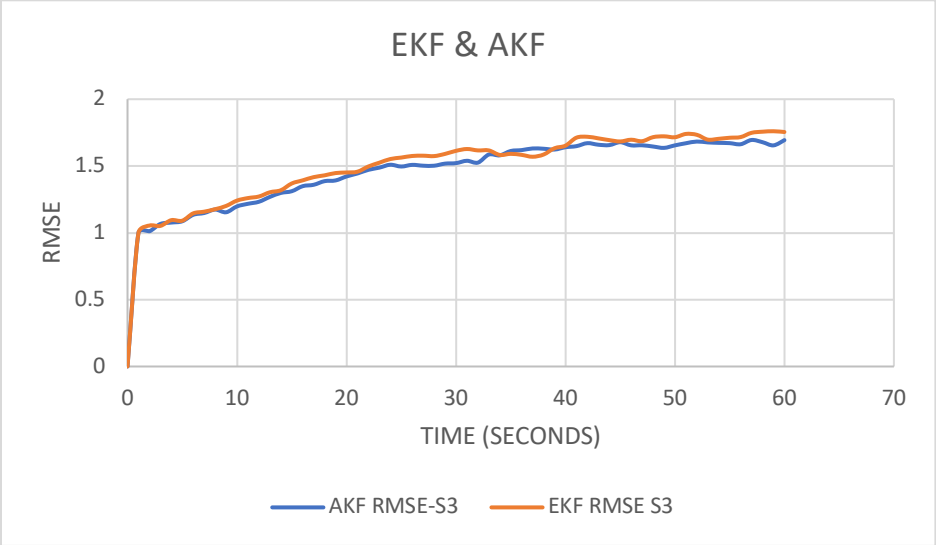

Fig.12

6.2 Effect of unknown measurement noise covariance

When noise covariance is unknown or not well defined its uncertainty increases, the filter may either overestimate or underestimate the noise level. Overestimating noise can make the filter overly conservative, reacting slowly to changes in the system. Conversely, underestimating noise can make the filter overly sensitive, causing it to react to random noise as if it were real data. Both scenarios degrade the filter's accuracy and reliability.

Benefits of adaptive filtering is that in the absence of known noise covariance, an AKF's ability to adjust noise levels dynamically offers a clear advantage by observing the structure of incoming data, the AKF can approximate noise levels in real time. This adaptability makes the AKF better suited for applications where noise characteristics are unpredictable.

Now we will study some cases where we will see how the unknown measurement noise covariance is affecting with the help of some plots-

For 300 Monte Carlo runs:

- $R = [0.1]$ & $R_f = [0.3]$

STATE 1:

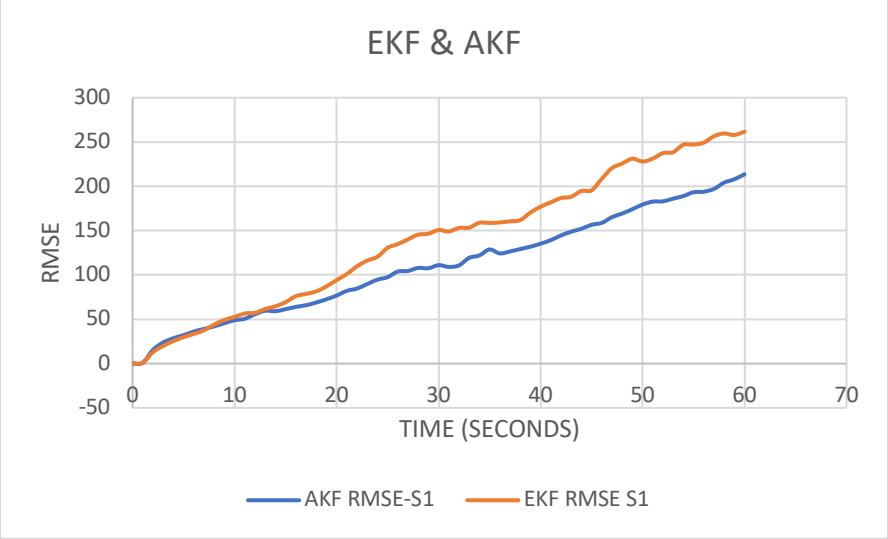

Fig.13

STATE 2:

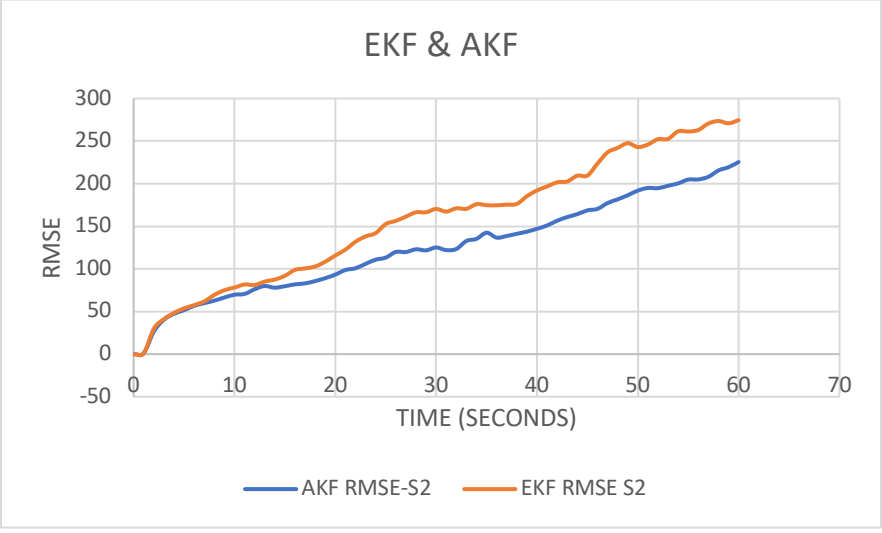

Fig.14

STATE 3:

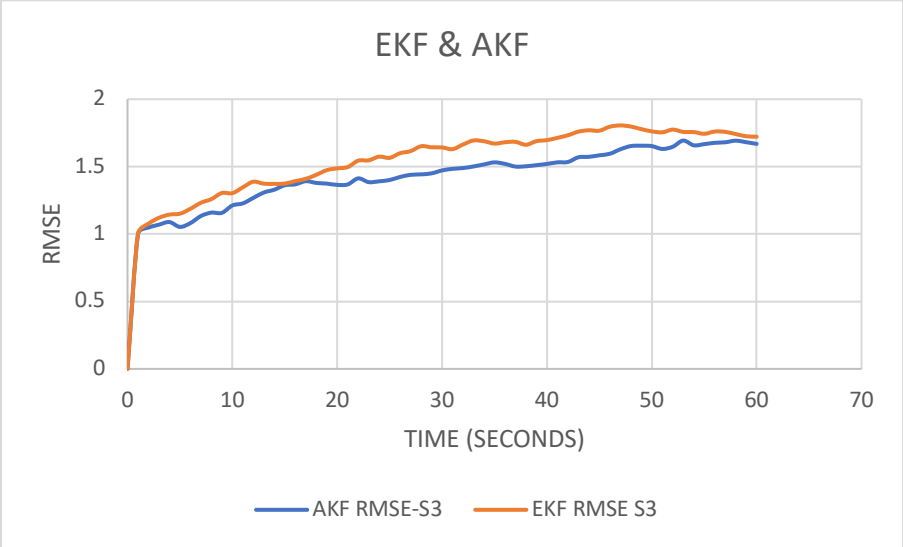

Fig.15

❖ $R = [0.1]$ & $R_f = [0.4]$

STATE 1:

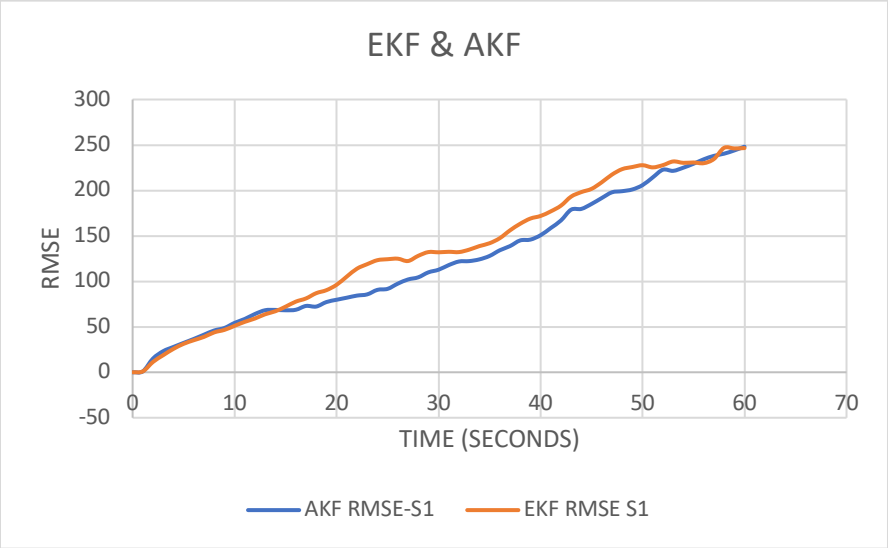

Fig.16

STATE 2:

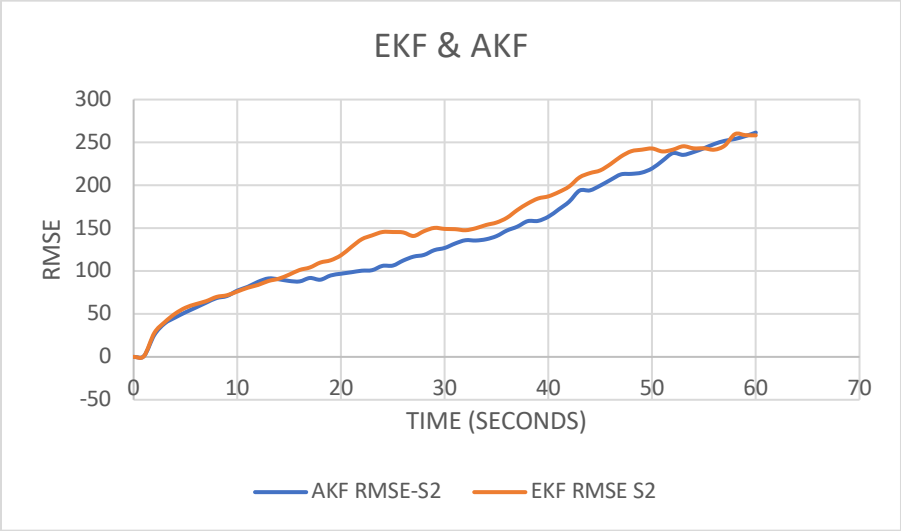

Fig.17

STATE 3:

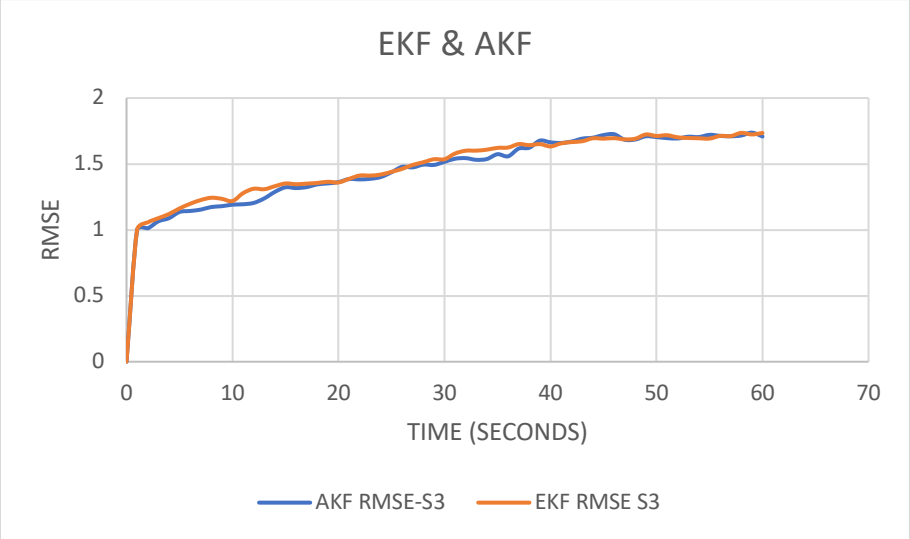

Fig.18

❖ $R = [0.1]$ & $R_f = [0.5]$

STATE 1:

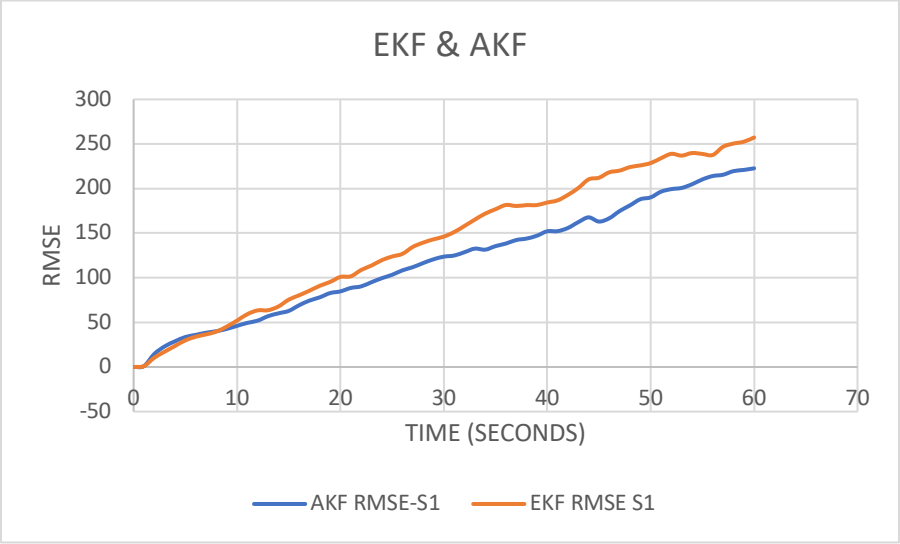

Fig.19

STATE 2:

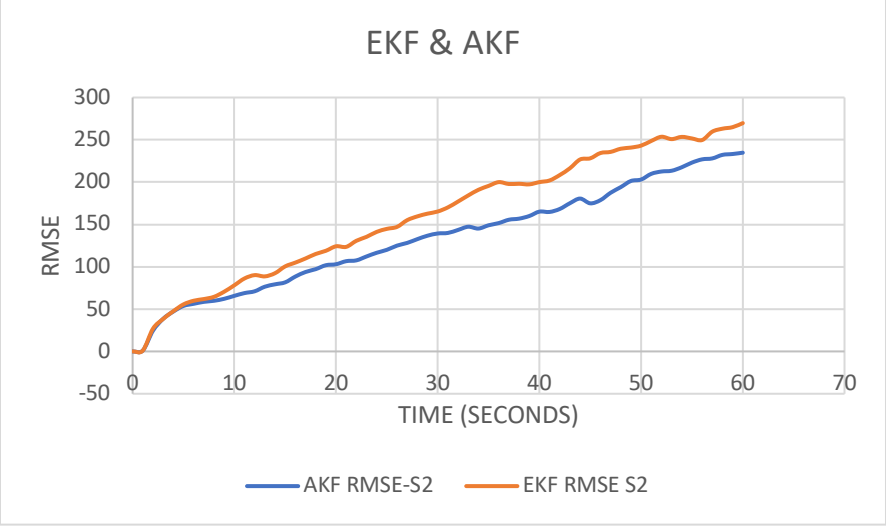

Fig.20

STATE 3:

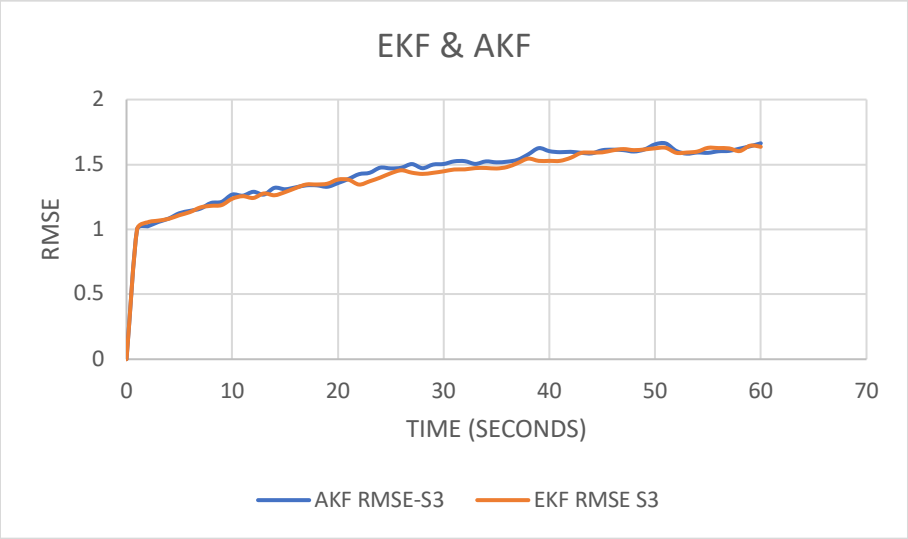

Fig.21

❖ $R = [0.1]$ & $R_f = [1.0]$

STATE 1:

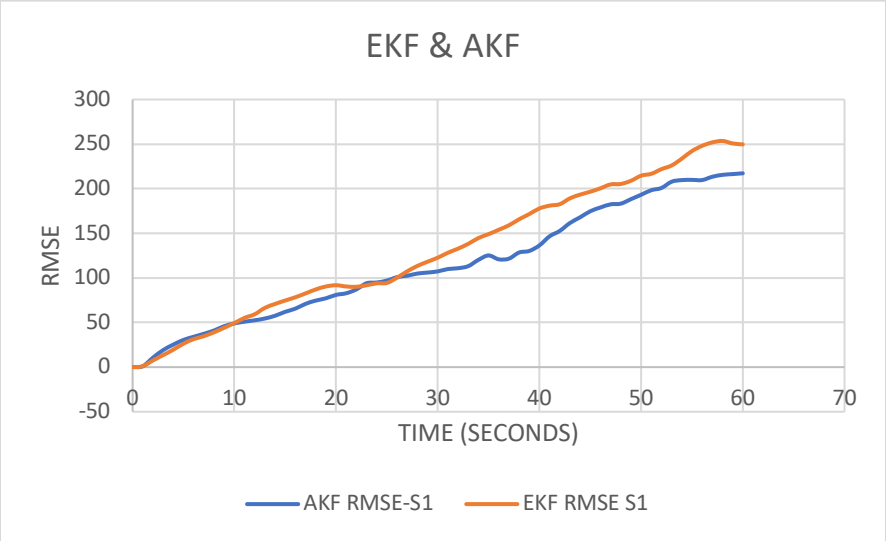

Fig.22

STATE 2:

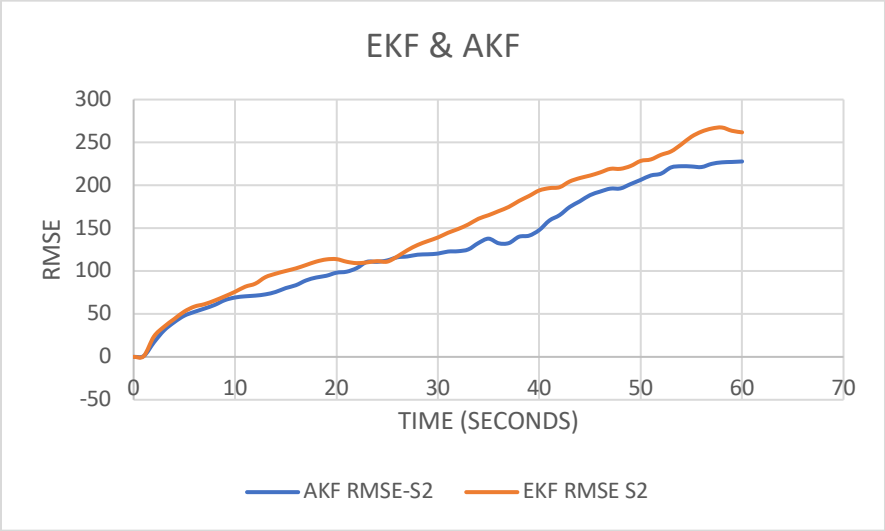

Fig.23

STATE 3:

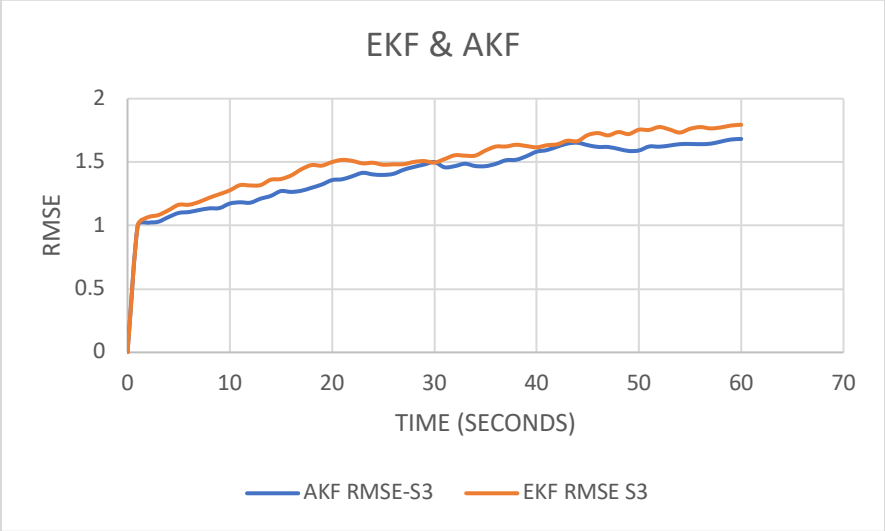

Fig.24

Chapter 7 – Discussions & conclusions

This thesis has presented a study of estimating the states for linear and nonlinear systems through the application of the Kalman Filter (KF), Extended Kalman Filter (EKF), and recently proposed Adaptive Kalman Filter (AKF). In particular, we focused on a comparative evaluation between the EKF and the recently proposed AKF for a specified nonlinear system. Through this examination, we aimed to understand how each filter adapts to changing system dynamics, especially under varying levels of measurement noise covariance.

The EKF, with its ability to linearize nonlinear systems around a current estimate, has proven effective in handling systems with mild nonlinearity. Our findings confirm that when the measurement noise covariance R is known, the EKF provides reliable estimates with minimal error. However, as system nonlinearity intensifies or measurement noise becomes inconsistent, the EKF struggles to maintain accuracy, as its performance relies on fixed assumptions about R and noise structure.

The AKF, on the other hand, incorporates an adaptive approach to dynamically adjust its process noise covariance based on the observed structure of measurements. This adaptability proved advantageous, especially in scenarios where the measurement noise covariance R was unknown. Our results highlight that, in the absence of a known R, the AKF significantly outperforms the EKF, showcasing its robustness in uncertain and changing environments.

This thesis concludes that the choice between the EKF and AKF is significantly influenced by the knowledge of the measurement noise covariance. When R is known and consistent, both filters deliver similar levels of accuracy, with the EKF being a suitable, less complex option for mildly nonlinear systems. However, when R is unknown or fluctuates, the AKF's adaptive noise tuning provides a distinct advantage, as it can self-adjust to maintain accuracy and prevent divergence.

These findings underscore the importance of adaptive filtering in environments where noise parameters cannot be reliably predefined, as is often the case in applications like autonomous navigation, robotic systems, or real-time monitoring of nonlinear processes. The AKF's flexibility to unknown noise characteristics and fluctuating measurement quality positions it as a valuable tool for state estimation in complex, real-world scenarios where traditional filters may fall short.

In summary, while the EKF is advantageous for stable and predictable environments with mild nonlinearity, the AKF demonstrates superior performance under uncertain conditions, particularly when the noise covariance cannot be easily ascertained. This research adds to the body of knowledge on Kalman filtering for nonlinear systems and provides a foundation for further developments in adaptive filtering techniques.

Chapter 8 – Future scope

Several avenues for future work emerge from this thesis. First, there is potential to optimize the computational efficiency of the AKF algorithm. The current implementation, while effective, could be improved to reduce computational burden and decrease run time, especially for high-volume Monte Carlo simulations. Enhancing algorithmic efficiency will be crucial for real-time applications that require rapid processing and low-latency responses. Additionally, exploring further optimizations of the AKF algorithm itself may allow it to better handle severe nonlinearities, expanding its applicability to more complex nonlinear systems where the EKF is insufficient.

There is also scope to test the AKF in real-world applications beyond simulated systems, such as autonomous vehicle navigation, UAV control, or battery state-of-charge estimation in fluctuating operational conditions. These applications would not only validate the AKF's adaptability under real conditions but also provide insights into further improvements for practical, industry-scale implementations.

In conclusion, the AKF holds considerable promise for adaptive state estimation, and by addressing the optimization and real-world application challenges highlighted here, future research can make significant strides in advancing adaptive filtering technologies for nonlinear systems.

Chapter 9 – References

1. [Brown2012] R. G. Brown and P. Y. C. Hwang, *Introduction to Random Signals and Applied Kalman Filtering: with MATLAB Exercises*, 4th ed. Hoboken, NJ: John Wiley & Sons, 2012.

2. [Das2014] M. Das, S. Sadhu and T. K. Ghoshal, "*Spacecraft attitude & rate estimation by an adaptive unscented Kalman filter*," Proceedings of The 2014 International Conference on Control, Instrumentation, Energy and Communication (CIEC), Calcutta, India, 2014, pp. 46-50, doi: 10.1109/CIEC.2014.6959047.

3. [Das2015] M. Das, A. Dey, S. Sadhu and T. K. Ghoshal, "*Adaptive Unscented Kalman Filter at the presence of non-additive measurement noise*," 2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO), Colmar, France, 2015, pp. 614-620.

4. [Shrivastava2019] P. Shrivastava, T. K. Soon, M. Y. I. B. Idris, and S. Mekhilef, "*Overview of model-based online state-of-charge estimation using Kalman filter family for lithium-ion batteries*," *Renewable and Sustainable Energy Reviews*, vol. 113, no. 109233, pp. 1-14, Jun. 2019, doi: 10.1016/j.rser.2019.06.040.

5. [Proctor2020] M. Proctor, N. Tian and H. Fang, "*State-of-Charge Estimation for Batteries Based on the Nonlinear Double-Capacitor Model and Extended Kalman Filter*," 2020 IEEE Green Technologies Conference (Green Tech), Oklahoma City, OK, USA, 2020, pp. 10-15, doi: 10.1109/GreenTech46478.2020.9289704.

6. [Ilies2020] A. I. Ilieș, G. Chindriș and D. Pitică, "*A Comparison between State of Charge Estimation Methods: Extended Kalman Filter and Unscented Kalman Filter*," 2020 IEEE 26th International Symposium for Design and Technology in Electronic Packaging (SIITME), Pitesti, Romania, 2020, pp. 376-381, doi: 10.1109/SIITME50350.2020.9292232.

7. [Takayama2024] Y. Takayama, T. Urakubo and H. Tamaki, "*Adaptive Choice of Process Noise Covariance in Kalman Filter Using Measurement Matrices*," in IEEE Transactions on Control Systems Technology, vol. 32, no. 3, pp. 934-944, May 2024, doi: 10.1109/TCST.2023.3339732.

Appendices

1. **MATLAB CODE FOR EKF**

```
clc;
clear all;
close all;
syms s t;

%-------------------------------------------------------------------------------------------%
%Step 1 = Matrix calculations
Cb=10037;
Cs=973;
Rb=0.019;
Rs=0;
R1=0.02;
C1=3250;

a11=-1/(Cb*(Rb+Rs));
a12=1/(Cb*(Rb+Rs));
a13=0;
a21=1/(Cs*(Rb+Rs));
a22=-1/(Cs*(Rb+Rs));
a23=0;
a31=0;
a32=0;
a33=-1/(R1*C1);

A=[a11 a12 a13;
  a21 a22 a23;
  a31 a32 a33];

b11=Rs/(Cs*(Rb+Rs));
b21=Rb/(Cs*(Rb+Rs));
b31=-1/C1;

B=[b11;
  b21;
  b31];

%-----------------------------%
```

```
% Taking the Inverse Laplace transform and making the F matrix.

si=s*eye(3);
A_inverse=inv(si - A);
A_det=det(A);
F_tilda = ilaplace(A_inverse);

%-----------------------------%
% Performing integration of the F matrix we have to find the G matrix.

f11=(@(t)(55882864600102721657*exp(-
(632343616903526209657*t)/106570299278333900226 56))/632343616903526209657 + 576460752303423488000/632343616903526209657);
f12=(@(t)55882864600102721657/632343616903526209657 -
(55882864600102721657*exp(-
(632343616903526209657*t)/106570299278333900226 56))/632343616903526209657);

f21=(@(t)576460752303423488000/632343616903526209657 -
(576460752303423488000*exp(-
(632343616903526209657*t)/106570299278333900226 56))/632343616903526209657);
f22=(@(t)(576460752303423488000*exp(-
(632343616903526209657*t)/106570299278333900226 56))/632343616903526209657 + 55882864600102721657/632343616903526209657);
f33=(@(t)exp(-t/65));

i1=integral(f11,0,1);
i2=integral(f12,0,1);
i3=0;
i4=integral(f21,0,1);
i5=integral(f22,0,1);
i6=0;
i7=0;
i8=0;
i9=integral(f33,0,1);

G_bar=[i1 i2 i3;i4 i5 i6;i7 i8 i9];
G= G_bar*B;


t=1;
f11=55882864600102721657*exp(-
(632343616903526209657*t)/106570299278333900226 56)/632343616903526209657
+ 576460752303423488000/632343616903526209657;
```

```
f12=(55882864600102721657/632343616903526209657 -
(55882864600102721657*exp(-
(632343616903526209657*t)/106570299278333900226 56))/632343616903526209657);
f13=(0);
f21=(576460752303423488000/632343616903526209657 -
(576460752303423488000*exp(-
(632343616903526209657*t)/106570299278333900226 56))/632343616903526209657);
f22=((576460752303423488000*exp(-
(632343616903526209657*t)/106570299278333900226 56))/632343616903526209657 + 55882864600102721657/632343616903526209657);
f23=(0);
f31=(0);
f32=(0);
f33=exp(-t/65);

F_f=[f11 f12 f13;f21 f22 f23;f31 f32 f33];

%-----------------------------%
% Finding the "H_f" via making the Jacobian
Vs = 0.1;
G;
F_f;

%-------------------------------------------------------------------------------------------%
%Step 2 = Truth parameters initializations
q11 = 0.1; q12 = 0; q13 = 0;
q21 = 0; q22 = 0.1; q23 = 0;
q31 = 0; q32 = 0; q33 = 0.1;

q(1,1) = 0.1; q(1,2)  = 0; q(1,3)  = 0;
q(2,1)  = 0; q(2,2)  = 0.1; q(2,3)  = 0;
q(3,1)  = 0; q(3,2)  = 0; q(3,3)  = 0.1;

F=F_f;
Q=[q(1,1) q(1,2) q(1,3);
    q(2,1) q(2,2) q(2,3);
    q(3,1) q(3,2) q(3,3)];
R=[0.1];

%-------------------------------------------------------------------------------------------%
%Step 3=Filter parameters
G;
F_f;
Q_f=[q(1,1) q(1,2) q(1,3);
```

```
    q(2,1) q(2,2) q(2,3);
    q(3,1) q(3,2) q(3,3)];
R_f=[0.5];
%-------------------------------------------------------------------------------------------%
order = 3;
N = 60;
MCruns = 300;
limit = MCruns*N;

xt=zeros(order,1); %state
x_minus=zeros(order,1); %previous estimate
x=zeros(order,1); %estimates
xhat=zeros(order,1);
p=zeros(order,order); %covariance
p_minus=zeros(order,order); %previous covariance
xstore=zeros(limit,1);

Xstore=zeros(order,1);
Xhatstore=zeros(order,1);
ystore=zeros(1,1);

y=zeros(1,1);
e=zeros(order,1);
z=zeros(1,1);%y(j,1)-H_f*x_minus(j,1)
temp=zeros(1,1);
K=zeros(order,1);

pnoise = zeros(1,N);
mnoise = zeros(1,N);

I = eye(order,order);

%-------------------------------------------------------------------------------------------%

for j = 1:MCruns %Mcruns(:,:,j,i)
x(:,:,j,1)=[1;1;1];
  for i = 1:N %time (:,:,j,i)
    for h=1:order
      pnoise = zeros(1,N);
      pnoise = pnoise + sqrt(q(h,h)) * randn(1,N);
      w(h,:) = pnoise;
    end

    for i = 2:N
      u=2;
      x(:,:,j,i) = F * x(:,:,j,i-1) + G*u + w(:,i-1);
```

```
      end
    end
end
  for j = 1:MCruns
   for temp=1:limit
   for i = 1:N

      for h=1:order
        v = mnoise + sqrt(R)* randn(1,N);
      end

      for i=2:N
           y(:,:,j,i) = 3.2 + (2.59 * x(2,:,j,i)) - ((9.003*x(2,:,j,i)).^2) + v(:,i-1);
      end
   end
   end
  end

for j = 1:MCruns

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

    for i = 1
     partial_deriv_H = 2.59 - 18.006 * xhat(2,i) ;

     H_f = [0 partial_deriv_H 0];

     p_minus(:,:,j,i) = F_f * p(:,:,j,i) * F_f' + Q_f; % for projecting ahead

     K(:,:,j,i) = p_minus(:,:,j,i) * H_f' / (H_f * p_minus(:,:,j,i) * H_f' + R_f); % kalman gain

     xhat_minus(:,:,j,i) = F_f * xhat(:,:,j,i); % for projecting ahead

     z(j,i) =  y(:,:,j,i) - H_f * xhat_minus(:,:,j,i);

     xhat(:,:,j,i) = xhat_minus(:,:,j,i) + K(:,:,j,i) * z(j,i); %updating the estimate

    p(:,:,j,i+1) = p_minus(:,:,j,i) - K(:,:,j,i) *(H_f * p_minus(:,:,j,i) * H_f' + R_f)*K(:,:,j,i)';

    e(:,:,j,i) = x(:,:,j,i) - xhat(:,:,j,i); %updating the error

     for i = 2:N

     partial_deriv_H = 2.59 - 18.006 * xhat(2,i-1) ;
```

```
    H_f = [0 partial_deriv_H 0];

    p_minus(:,:,j,i) = F_f * p(:,:,j,i) * F_f' + Q_f; % for projecting ahead

    K(:,:,j,i) = p_minus(:,:,j,i) * H_f' / (H_f * p_minus(:,:,j,i) * H_f' + R_f); % kalman gain

    xhat_minus(:,:,j,i) = F_f * xhat(:,:,j,i-1); % for projecting ahead

    z(j,i) = y(:,:,j,i) - H_f * xhat_minus(:,:,j,i);

    xhat(:,:,j,i) = xhat_minus(:,:,j,i) + K(:,:,j,i) * z(j,i); %updating the estimate

   p(:,:,j,i+1) = p_minus(:,:,j,i) - K(:,:,j,i) *(H_f * p_minus(:,:,j,i) * H_f' + R_f)*K(:,:,j,i)';

   e(:,:,j,i) = x(:,:,j,i) - xhat(:,:,j,i); %updating the error

     end
   end
 end

for j=1:MCruns
   for i=1:N
      for k=1:order

      P(j,i,k)=p(k,k,j,i);
      E(j,i,k)=e(k,1,j,i); %storing the errors.
      X(j,i,k)=x(k,1,j,i);
      Xhat(j,i,k)=xhat(k,1,j,i);
   end
   end
 end
%-------------------------------------------------------------------%
 % To store the error of each state separately

for j=1:MCruns
   for i=1:N
      for k=1:order
      E1(j,i)=E(j,i,1);
      E2(j,i)=E(j,i,2);
      E3(j,i)=E(j,i,3);
      x1(j,i)=X(j,i,1);
      x2(j,i)=X(j,i,2);
      x3(j,i)=X(j,i,3);
      xhat1(j,i)=Xhat(j,i,1);
      xhat2(j,i)=Xhat(j,i,2);
      xhat3(j,i)=Xhat(j,i,3);
```

```
   end
  end
end
%-------------------------------------------------------------------%
 % To store the error-square of each state separately
for j=1:MCruns
   for i=1:N
      for k=1:order
      E1square(j,i)=E1(j,i).^2;
      E2square(j,i)=E2(j,i).^2;
      E3square(j,i)=E3(j,i).^2;
   end
  end
end
 %-------------------------------------------------------------------%
 % To store the mean-square-error of each state separately
for j=1:MCruns
   for i=1:N
      for k=1:order
      E1squaremean=mean(E1square);
      E2squaremean=mean(E2square);
      E3squaremean=mean(E3square);
   end
  end
end
 %-------------------------------------------------------------------%
% To store the mean-square-error of each state separately
for j=1:MCruns
   for i=1:N
      for k=1:order
      E1rms=sqrt(E1squaremean);
      E2rms=sqrt(E2squaremean);
      E3rms=sqrt(E3squaremean);
   end
  end
end
%-------------------------------------------------------------------%
for k=1:order
    for i=1:N
     Pstore(k,i)=P(1,i,k);
    end
 end
 for k=1:order
  for i=1:N
    for j=1:MCruns
       Xstore(:,:,i,j)=x(:,:,j,i);
```

```
      Xhatstore(:,:,i,j)=xhat(:,:,j,i);
    end
 end
 end
%-------------------------------------------------------------------------------------------------%
% NOW WE SHALL PLOT BELOW %
 for i=1:order
   esquare(:,:,i)=E(:,:,i).^2;
   mse(i,:)=mean(esquare(:,:,i));
   g = 1:N;
   figure(i)
   subplot(3,1,1);
   hold on;
   grid on;
   plot(g,mse(i,:),'r','linewidth',1.5)
   plot(g,Pstore(i,g),'b','linewidth',1.5)
   legend('mean square error')
   title(['EKF Plot for state ' ,num2str(i), ' having ' ,num2str(MCruns), ' MCruns']);
   xlabel(['Time Step = ' ,num2str(N),  ' seconds']);
end
% %
for i=1:order
   rmse(i,:)=sqrt(mse(i,:));
   g = 1:N;
   figure(i)
   subplot(3,1,2);
   hold on;
   grid on;
   plot(g,rmse(i,:),'r','linewidth',1.5)
   legend('root mean square error')
   title(['EKF Plot for state ' ,num2str(i), ' having ' ,num2str(MCruns), ' MCruns']);
   xlabel(['Time Step = ' ,num2str(N),  ' seconds']);
end
%%----------------------%%
for i=1:order
   g = 1:N;
   figure(i)
   subplot(3,1,3);
   hold on;
   grid on;
   plot(g,Xstore(i,g),'r','linewidth',1.5)
   plot(g,Xhatstore(i,g),'b','linewidth',1.5)
   legend('x ','x-hat')
   title(['EKF Plot for state ' ,num2str(i), ' having ' ,num2str(MCruns), ' MCruns']);
   xlabel(['Time Step = ' ,num2str(N),  ' seconds']);
end
```

2. **MATLAB CODE FOR AKF**

```
clc;
clear all;
close all;
syms s t;

%-----------------------------------------------------------------------------------------%
%Step 1 = Matrix calculations
Cb=10037;
Cs=973;
Rb=0.019;
Rs=0;
R1=0.02;
C1=3250;

a11=-1/(Cb*(Rb+Rs));
a12=1/(Cb*(Rb+Rs));
a13=0;
a21=1/(Cs*(Rb+Rs));
a22=-1/(Cs*(Rb+Rs));
a23=0;
a31=0;
a32=0;
a33=-1/(R1*C1);

A=[a11 a12 a13;
  a21 a22 a23;
  a31 a32 a33];

b11=Rs/(Cs*(Rb+Rs));
b21=Rb/(Cs*(Rb+Rs));
b31=-1/C1;

B=[b11;
  b21;
  b31];

%-----------------------------%
% Taking the Inverse Laplace transform and making the F matrix.

si=s*eye(3);
A_inverse=inv(si - A);
A_det=det(A);
F_tilda = ilaplace(A_inverse);
```

```
%-----------------------------%
% Performing integration of the F matrix we have to find the G matrix.

f11=(@(t)(55882864600102721657*exp(-
(6323436169035262096 57*t)/106570299278333900226 56))/6323436169035262096
7 + 5764607523034234880006323436169035262096 57);
f12=(@(t)55882864600102721657/6323436169035262096 57 -
(55882864600102721657*exp(-
(6323436169035262096 57*t)/106570299278333900226 56))/6323436169035262096
7);
f21=(@(t)57646075230342348800006323436169035262096 57 -
(57646075230342348800 0*exp(-
(6323436169035262096 57*t)/106570299278333900226 56))/6323436169035262096
7);
f22=(@(t)(57646075230342348800 0*exp(-
(6323436169035262096 57*t)/106570299278333900226 56))/6323436169035262096
7 + 55882864600102721657/6323436169035262096 57);
f33=(@(t)exp(-t/65));

i1=integral(f11,0,1);
i2=integral(f12,0,1);
i3=0;
i4=integral(f21,0,1);
i5=integral(f22,0,1);
i6=0;
i7=0;
i8=0;
i9=integral(f33,0,1);

G_bar=[i1 i2 i3;i4 i5 i6;i7 i8 i9];
G= G_bar * B;

t=1;
f11=55882864600102721657*exp(-
(6323436169035262096 57*t)/106570299278333900226 56)/6323436169035262096 57
+ 5764607523034234880006323436169035262096 57;
f12=(55882864600102721657/6323436169035262096 57 -
(55882864600102721657*exp(-
(6323436169035262096 57*t)/106570299278333900226 56))/6323436169035262096
7);
f13=(0);
f21=(57646075230342348800 06323436169035262096 57 -
(57646075230342348800 0*exp(-
(6323436169035262096 57*t)/106570299278333900226 56))/6323436169035262096
7);
```

```
f22=((576460752303423488000*exp(-
(6323436169035262096 57*t)/106570299278333900226 56))/6323436169035262096 57 + 558828646001027216 57/6323436169035262096 57);
f23=(0);
f31=(0);
f32=(0);
f33=exp(-t/65);

F_f=[f11 f12 f13;f21 f22 f23;f31 f32 f33]
G;
F_f;
%-------------------------------------------------------------------------------------------%

%Step 2 = Truth parameters initializations
q11 = 0.1; q12 = 0; q13 = 0;
q21 = 0; q22 = 0.1; q23 = 0;
q31 = 0; q32 = 0; q33 = 0.1;

q(1,1) = 0.1; q(1,2)  = 0; q(1,3)  = 0;
q(2,1)  = 0; q(2,2)  = 0.1; q(2,3)  = 0;
q(3,1)  = 0; q(3,2)  = 0; q(3,3)  = 0.1;

F=F_f;
Q=[q(1,1) q(1,2) q(1,3);
     q(2,1) q(2,2) q(2,3);
     q(3,1) q(3,2) q(3,3)];
R=[0.1];

%-------------------------------------------------------------------------------------------%
%Step 3=Filter parameters
F_f;
Q_f=  [q(1,1) q(1,2) q(1,3);
     q(2,1) q(2,2) q(2,3);
     q(3,1) q(3,2) q(3,3)];
R_f=[0.4];


%-------------------------------------------------------------------------------------------%
order = 3;
N = 10;
MCruns = 10;
limit = MCruns * N;

xt=zeros(order,1); %state
x_minus=zeros(order,1); %previous estimate
xhat_minus=zeros(order,1);
```

```
x=zeros(order,1); %estimates
xhat=zeros(order,1);
p=zeros(order,order); %covariance
p_minus=zeros(order,order); %previous covariance
xstore=zeros(limit,1);

Xstore=zeros(order,1);
Xhatstore=zeros(order,1);
ystore=zeros(1,1);

O_k=zeros(order,order);
alpha_k_star=zeros(1,1);
delta_pk_start=zeros(order,order);
delta_kk_star=zeros(order,1);
delta_qk_star=zeros(order,order);
kk_delta_qk_star=zeros(order,1);

y=zeros(1,1);
e=zeros(order,1);
z=zeros(1,1);%y(j,1)-H_f*x_minus(j,1)
temp=zeros(1,1);
K=zeros(order,1);

pnoise = zeros(1,N);
mnoise = zeros(1,N);

I = eye(order,order);

%-------------------------------------------------------------------------------------------%

for j = 1:MCruns %Mcruns(:,:,j,i)
x(:,:,j,1)=[1;1;1];
   for i = 1:N %time (:,:,j,i)
      for h=1:order
        pnoise = zeros(1,N);
        pnoise = pnoise + sqrt(q(h,h)) * randn(1,N);
        w(h,:) = pnoise;
       end

     for i = 2:N
        u = 2;
        x(:,:,j,i) = F * x(:,:,j,i-1) + G*u + w(:,i-1);
     end
   end
end
```

```
  for j = 1:MCruns
   for temp=1:limit
   for i = 1:N

      for h=1:order
        v = mnoise + sqrt(R)* randn(1,N);
      end

      for i=2:N

          y(:,:,j,i) = 3.2 + (2.59 * x(2,:,j,i)) - ((9.003*x(2,:,j,i)).^2) + v(:,i-1);
    end
  end
 end
  end

 for j = 1:MCruns

    xhat(:,:,j,1)=[0;0;0];
    p(:,:,j,1)=[1 0 0;0 1 0;0 0 1];
    delta_qk_star(:,:,j,i)=[0 0 0;0 0 0;0 0 0];

    for i = 1

     partial_deriv_H = 2.59 - 18.006 * xhat(2,i) ;

     H_f = [0 partial_deriv_H 0];

     p_minus(:,:,j,i) = F_f * p(:,:,j,i) * F_f' + Q_f + delta_qk_star(:,:,j,i); % for projecting ahead

     K(:,:,j,i) = p_minus(:,:,j,i) * H_f' / (H_f * p_minus(:,:,j,i) * H_f' + R_f); % kalman gain

     xhat_minus(:,:,j,i) = F_f * xhat(:,:,j,i); % for projecting ahead

     O_k(:,:,j,i) = H_f' * inv(R_f) * H_f; % Note : here O_k is 3by3

     alpha_k_star(j,i) = trace ((O_k(:,:,j,i).^2) / (trace((((O_k(:,:,j,i)).^5) * p_minus(:,:,j,i)) +
(O_k(:,:,j,i)).^4))); %note : here alpha_k_star(j,i) is 1 by1

     delta_pk_start(:,:,j,i) =  alpha_k_star(j,i) * O_k(:,:,j,i);%note - delta_pk_start is 3by3

     delta_kk_star(:,:,j,i) = delta_pk_start(:,:,j,i) * H_f' * inv(R_f);%note - delta_kk_star is 3by1

     delta_qk_star(:,:,j,i) = inv(I -  (K(:,:,j,i) + delta_kk_star(:,:,j,i)) * H_f) * (delta_pk_start(:,:,j,i)
- delta_kk_star(:,:,j,i) * (H_f * p_minus(:,:,j,i) * H_f' + R_f) * delta_kk_star(:,:,j,i)') * inv((I -
(K(:,:,j,i) + delta_kk_star(:,:,j,i)) * H_f))';
```

```
kk_delta_qk_star(:,:,j,i) = K(:,:,j,i) + delta_kk_star(:,:,j,i);

z(j,i) =  y(:,:,j,i) - H_f * xhat_minus(:,:,j,i);

xhat(:,:,j,i) = xhat_minus(:,:,j,i) + kk_delta_qk_star(:,:,j,i) * z(j,i);

p(:,:,j,i+1) = (I - (kk_delta_qk_star(:,:,j,i) * H_f)) * p_minus(:,:,j,i);

e(:,:,j,i) = x(:,:,j,i) - xhat(:,:,j,i); %updating the error

for i = 2:N

partial_deriv_H = 2.59 - 18.006 * xhat(2,i-1) ;

H_f = [0 partial_deriv_H 0];

p_minus(:,:,j,i) = F_f * p(:,:,j,i) * F_f' + Q_f + delta_qk_star(:,:,j,i); % for projecting ahead

K(:,:,j,i) = p_minus(:,:,j,i) * H_f' / (H_f * p_minus(:,:,j,i) * H_f' + R_f); % kalman gain

xhat_minus(:,:,j,i) = F_f * xhat(:,:,j,i-1); % for projecting ahead

O_k(:,:,j,i) = H_f' * inv(R_f) * H_f; % Note : here O_k is 3by3

alpha_k_star(j,i) = trace ((O_k(:,:,j,i).^2) / (trace((((O_k(:,:,j,i)).^5) * p_minus(:,:,j,i)) +
(O_k(:,:,j,i)).^4))); %note : here alpha_k_star(j,i) is 1 by1

delta_pk_start(:,:,j,i) =  alpha_k_star(j,i) * O_k(:,:,j,i);%note - delta_pk_start is 3by3

delta_kk_star(:,:,j,i) = delta_pk_start(:,:,j,i) * H_f' * inv(R_f);%note - delta_kk_star is 3by1

delta_qk_star(:,:,j,i) = inv(I -  (K(:,:,j,i) + delta_kk_star(:,:,j,i)) * H_f) * (delta_pk_start(:,:,j,i)
- delta_kk_star(:,:,j,i) * (H_f * p_minus(:,:,j,i) * H_f' + R_f)*delta_kk_star(:,:,j,i)') * inv((I -
(K(:,:,j,i) + delta_kk_star(:,:,j,i)) * H_f))';

kk_delta_qk_star(:,:,j,i) = p_minus(:,:,j,i) * H_f' * (inv(H_f * p_minus(:,:,j,i) * H_f' + R_f));

z(j,i) =  y(:,:,j,i) - H_f * xhat_minus(:,:,j,i);

xhat(:,:,j,i) = xhat_minus(:,:,j,i) + K(:,:,j,i) * z(j,i); %updating the estimate

p(:,:,j,i+1) = (I - (kk_delta_qk_star(:,:,j,i) * H_f)) * p_minus(:,:,j,i);

e(:,:,j,i) = x(:,:,j,i) - xhat(:,:,j,i); %updating the error
```

```
    end
  end
end
%-------------------------------------------------------------------%
% Storing the error and P matrix in the correct order%

for j=1:MCruns
  for i=1:N
    for k=1:order

    P(j,i,k)=p(k,k,j,i);
    E(j,i,k)=e(k,1,j,i); %storing the errors.
    X(j,i,k)=x(k,1,j,i);
    Xhat(j,i,k)=xhat(k,1,j,i);
  end
  end
end

%-------------------------------------------------------------------%
% To store the error of each state separately

for j=1:MCruns
  for i=1:N
    for k=1:order
    E1(j,i)=E(j,i,1);
    E2(j,i)=E(j,i,2);
    E3(j,i)=E(j,i,3);

    x1(j,i)=X(j,i,1);
    x2(j,i)=X(j,i,2);
    x3(j,i)=X(j,i,3);

    xhat1(j,i)=Xhat(j,i,1);
    xhat2(j,i)=Xhat(j,i,2);
    xhat3(j,i)=Xhat(j,i,3);
  end
 end
end
%-------------------------------------------------------------------%
% To store the error-square of each state separately
for j=1:MCruns
  for i=1:N
    for k=1:order
    E1square(j,i)=E1(j,i).^2;
    E2square(j,i)=E2(j,i).^2;
    E3square(j,i)=E3(j,i).^2;
```

```
   end
  end
end
%-------------------------------------------------------------------%
 % To store the mean-square-error of each state separately
for j=1:MCruns
   for i=1:N
      for k=1:order
      E1squaremean=mean(E1square);
      E2squaremean=mean(E2square);
      E3squaremean=mean(E3square);
   end
  end
end
%-------------------------------------------------------------------%
 % To store the mean-square-error of each state separately
for j=1:MCruns
   for i=1:N
      for k=1:order
      E1rms=sqrt(E1squaremean);
      E2rms=sqrt(E2squaremean);
      E3rms=sqrt(E3squaremean);
   end
  end
end
%-------------------------------------------------------------------%
 for k=1:order
    for i=1:N
     Pstore(k,i)=P(1,i,k);
    end
 end
 for k=1:order
  for i=1:N
    for j=1:MCruns
       Xstore(:,:,i,j)=x(:,:,j,i);
       Xhatstore(:,:,i,j)=xhat(:,:,j,i);
    end
 end
 end
%-------------------------------------------------------------------------------------------%
% NOW WE SHALL PLOT BELOW %
 for i=1:order
   esquare(:,:,i)=E(:,:,i).^2;
   mse(i,:)=mean(esquare(:,:,i));
   g = 1:N;
   figure(i)
```

```
   subplot(3,1,1);
   hold on;
   grid on;
   plot(g,mse(i,:),'r','linewidth',1.5)
   plot(g,Pstore(i,g),'b','linewidth',1.5)
   legend('mean square error')
   title(['AKF Plot for state ' ,num2str(i), ' having ' ,num2str(MCruns), ' MCruns']);
   xlabel(['Time Step = ' ,num2str(N),  ' seconds']);
end
% %
%
for i=1:order
   rmse(i,:)=sqrt(mse(i,:));
   g = 1:N;
   figure(i)
   subplot(3,1,2);
   hold on;
   grid on;
   plot(g,rmse(i,:),'r','linewidth',1.5)
   legend('root mean square error')
   title(['AKF Plot for state ' ,num2str(i), ' having ' ,num2str(MCruns), ' MCruns']);
   xlabel(['Time Step = ' ,num2str(N),  ' seconds']);
end
%%-----------------------%%
for i=1:order
   g = 1:N;
   figure(i)
   subplot(3,1,3);
   hold on;
   grid on;
   plot(g,Xstore(i,g),'r','linewidth',1.5)
   plot(g,Xhatstore(i,g),'b','linewidth',1.5)
   legend('x ','x-hat')
   title(['AKF Plot for state ' ,num2str(i), ' having ' ,num2str(MCruns), ' MCruns']);
   xlabel(['Time Step = ' ,num2str(N),  ' seconds']);
end
```